Proceedings of AUTOMATA 2010: 16th International workshop on cellular automata and discrete complex systems

Nazim A. Fatès, Jarkko Kari, Thomas Worsch

AUTOMATA

16th International Workshop on
Cellular Automata & Discrete Complex Systems

Proceedings

Edited by:
Nazim Fatès
Jarkko Kari
Thomas Worsch

Nancy, France
14-16 June
2010

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche NANCY - GRAND EST

Loria

Automata 2010

16^{th} International Workshop on Cellular Automata and Discrete Complex Systems

organised by INRIA Nancy - Grand Est
held at LORIA, Nancy
under the auspices of IFIP

June 14 – 16, 2010

Editors:

Nazim Fatès, Jarkko Kari and Thomas Worsch

Preface

This volume contains all the contributed papers presented at AUTOMATA 2010, the 16th international workshop on cellular automata and discrete complex systems. The workshop was held on June 14-16, 2010, at the LORIA laboratory in Nancy, France. AUTOMATA is an annual workshop on the fundamental aspects of cellular automata and related discrete dynamical systems. The spirit of the workshop is to foster collaborations and exchanges between researchers on these areas. The workshop series was started in 1995 by members of the Working Group 1.5 of IFIP, the International Federation for Information Processing.

The volume has two parts: Part I contains 9 full papers that were selected by a program committee from 21 submissions. These papers will also appear as proceedings volume **AL** of Discrete Mathematics and Theoretical Computer Science (DMTCS). The program committee consisted of 25 international experts on cellular automata and related models, and the selection was based on 2-4 peer reviews on each paper. Part II contains 18 short papers of work-in-progress and/or exploratory papers. Both paper categories combined, the workshop received 37 submissions.

Papers in this volume represent a rich sample of current research topics on cellular automata and related models. The papers include theoretical studies of the classical cellular automata model, but also many investigations into various variants and generalizations of the basic concept. The versatile nature and the flexibility of the model is evident from the presented papers, making it a rich source of new research problems for scientists representing a variety of disciplines.

In addition to the papers of this volume, the program of AUTOMATA 2010 contained four one-hour plenary lectures given by distinguished invited speakers:

- Enrico Formenti (University of Nice-Sophia Antipolis, France)
- Jean Mairesse (University of Paris 7, France)
- Ferdinand Peper (Himeji Institute of Technology, Japan)
- Guillaume Theyssier (University of Savoie, France)

The organisers are indebted to the invited speakers, who kindly accepted to pay part of their travel as a support to the conference.

The organizers gratefully acknowledge the support by the following institutions:

- European Society for Mathematical and Theoretical Biology (ESMTB)
- Rgion Lorraine and Mairie de Nancy
- Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA)
- Communauté urbaine Grand Nancy
- Nancy Université Henri Poincaré
- Nancy Université 2
- Nancy Université Institut national polytechnique de Lorraine

As the editors of these proceedings, we thank all contributors to the scientific program of the workshop. We are especially indebted to the invited speakers and the authors of the contributed papers. We would also like to thank the members of the Program Committee and the external reviewers of the papers.

May 28th, 2010

Nazim Fatès, Jarkko Kari, Thomas Worsch

Program Committee

Andy Adamatzky	University of the West of England, UK
Stefania Bandini	University of Milano - Bicocca, Italy
Pedro de Oliveira	Mackenzie Presbyterian University, Brasil
Andreas Deutsch	Dresden University of Technology, Germany
Nazim Fatès	INRIA Nancy Grand-Est, France, co-chair
Paola Flocchini	University of Ottawa, Chanada
Enrico Formenti	University of Nice-Sophia Antipolis, France
Henryk Fukś	Brock University, Canada
Jarkko Kari	University of Turku, Finland, co-chair
Martin Kutrib	University of Gießen, Germany
Anna Lawniczak	University of Guelph, Canada
Alejandro Maass	University of Chile, Chile
Danuta Makowiec	Gdansk University, Poland
Maurice Margenstern	University of Metz, France
Kenichi Morita	Hiroshima University, Japan
Nicolas Ollinger	University of Provence, France
Ferdinand Peper	National Institute of Information and Communications Technology, Japan
Juan Carlos Seck	Autonomous University of Hidalgo State, Mexico
Georgios Sirakoulis	Democritus University of Thrace, Greece
Klaus Sutner	Carnegie Mellon University, USA
Guillaume Theyssier	University of Savoie, France
Hiroshi Umeo	University of Osaka Electro-Communication, Japan
Laurent Vuillon	University of Savoie, France
Thomas Worsch	University of Karlsruhe, Germany
Jean-Baptiste Yunès	University of Paris 7, France

External Referees

Carsten Mente	University of Dresden, Germany
Niloy Ganguly	Indian Institute of Technology
Pierre Guillon	University of Turku, Finland
Andreas Malcher	University of Gießen, Germany
Charalampos Zinoviadis	University of Turku, Finland

Organizing Committee

Nicolas Alcaraz
Olivier Boure
Anne-Lise Charbonnier
Vincent Chevrier
Sylvain Contassot-Vivier
Nazim Fatès (chair)
Rachida Kasmi
Nikolaos Vlassopolous

Table of Contents

Author index

Faster Methods for Identifying Nontrivial Energy Conservation Functions for Cellular Automata

Leemon Baird[1], Barry Fagin[1]

[1]*Academy Center for Cyberspace Research, Department of Computer Science, US Air Force Academy, Colorado Springs, Colorado USA 80840*

The biggest obstacle to the efficient discovery of conserved energy functions for cellular auotmata is the elimination of the trivial functions from the solution space. Once this is accomplished, the identification of nontrivial conserved functions can be accomplished computationally through appropriate linear algebra.

As a means to this end, we introduce a general theory of trivial conserved functions. We consider the existence of nontrivial additive conserved energy functions ("nontrivials") for cellular automata in any number of dimensions, with any size of neighborhood, and with any number of cell states. We give the first known basis set for all trivial conserved functions in the general case, and use this to derive a number of optimizations for reducing time and memory for the discovery of nontrivials.

We report that the Game of Life has no nontrivials with energy windows of size 13 or smaller. Other 2D automata, however, do have nontrivials. We give the complete list of those functions for binary outer-totalistic automata with energy windows of size 9 or smaller, and discuss patterns we have observed.

Keywords: nontrivial conserved energy function, trivial conserved energy function, 1D cellular autamata, 2D cellular automata, Game of Life

1 Preliminaries: basic definitions

We consider cellular automata with k states in n dimensions. The *neighborhood* of a cellular automaton is the region of surrounding cells used to determine the next state of a given cell. The *window* of an energy function for a cellular automaton is the region of adjacent cells that contribute to the function. Both neighborhoods and windows are n-dimensional tensors, with the size of each dimension specified as a positive integer. Given the size of such a tensor, it is useful to define the following 3 sets of tensors.

Definition 1.1 *Cellular automata are composed of cells, each of which is in one of k states (or colors) at any given time. The set $\mathcal{C}$ is the set of such colors, and the set $\mathcal{C}_*$ is that set augmented with another color named *. (* denotes a special state with certain properties that simplify our proofs. It is explained in more detail in the pages that follow.)*

$$\mathcal{C} = \{0, 1, 2, \ldots, k-1\} \tag{1.1}$$

$$\mathcal{C}_* = \mathcal{C} \cup \{*\} \tag{1.2}$$

It is sometimes useful to choose one color to be treated specially. In all such cases, the color 0 will be chosen.

Definition 1.2 *An n-dimensional cellular automaton rule is a function R that gives the color of a given cell on the next time step as a function of a neighborhood of cells centered on that cell on the current time step. The neighborhood is an n-dimensional tensor of size $w_1 \times \cdots \times w_n$, where each w_i is an odd, positive integer.*

$$R : C^{w_1 \times \cdots \times w_n} \to \mathcal{C} \tag{1.3}$$

Definition 1.3 *An n-dimensional cellular automaton is an n-dimensional tensor whose elements are in $\mathcal{C}$, and which is updated on each time step according to a cellular automaton rule R, applied to every cell in parallel. The rule is a function applied to each cell and its neighbors, where neighbors wrap toroidally (i.e. the top edge is considered adjacent to the bottom, the left edge is adjacent to the right, and so on for each dimension).*

Definition 1.4 *The successor function advances a region within a cellular automaton one time step by applying a rule R to a region M of size $s_1 \times \cdots \times s_n$*

$$T : (\mathcal{C}^{w_1 \times \cdots \times w_n} \to \mathcal{C}) \times \mathcal{C}^{s_1 \times \cdots \times s_n} \to \mathcal{C}^{(s_1 - w_1 + 1) \times \cdots \times (s_n - w_n + 1)}$$

which is defined as:

$$T(R, M) = M' \text{ where } M'_{i_1,\ldots,i_n} = R(M_{(i_1 \ldots i_1 + w_1 - 1),\ldots,(i_n \ldots i_n + w_n - 1)}) \tag{1.4}$$

Note that $T(R, M)$ is defined for an M that is only a portion of the cells, and so it does not wrap around toroidally. Instead, it returns a tensor that is smaller than M in each dimension. Also note that the ellipses on the right side of the equation are used in two different ways. Each element of the result comes from applying the R function to only a portion of the M tensor, which includes those elements of M whose first coordinate is in the range $[i_1, i_1 + w_1 - 1]$, and whose second coordinate is in the range $[i_2, i_2 + w_2 - 1]$, and so on up to the nth coordinate being in the range $[i_n, i_n + w_n - 1]$.

Definition 1.5 *A linear additive energy function (or energy function) is a function $f : \mathcal{C}^{s_1 \times \cdots \times s_n} \to \mathbb{R}$ that assigns a real number to a window of size $s_1 \times \cdots \times s_n$ within a cellular automaton.*

Definition 1.6 *The total energy $e_{tot} : \mathcal{C}^{u_1 \times \cdots \times u_n} \to \mathbb{R}$ of a given state U of an entire cellular automaton universe with $u_1 \times \cdots \times u_n$ cells, with respect to a given energy function f, is*

$$e_{tot}(U) = \sum_W f(U_W) \tag{1.5}$$

where U is the universe state for a cellular automaton, W is the position of the energy window within that universe, and U_W is that window within the universe, which wraps toroidally at the edges of the universe.

Definition 1.7 *A conserved linear additive energy function (or a conserved function) for a given cellular automaton rule is an energy function that for a universe of any size, and for any given state of that universe, will assign the same total energy to that universe for both that state and its successor.*

Definition 1.8 *A trivial conserved linear additive energy function (or a trivial) is an energy function that for a universe of any size, will assign the same total energy to that universe regardless of its state. A nontrivial conserved linear additive energy function (or a nontrivial) for a given cellular automaton rule is a conserved energy function that is not trivial.*

Definition 1.9 *Given n positive integers $s_1, \ldots, s_n$ defining the size of an n-dimensional tensor, the set $\mathcal{B}(s_1, \ldots, s_n)$ is the set of all tensors over $\mathcal{C}$ of that size. This set is partitioned into two sets, $\mathcal{Z}(s_1, \ldots, s_n)$, the zero-sided tensors, which have at least one side that contains the origin element and is filled entirely with zero elements, and $\bar{\mathcal{Z}}(s_1, \ldots, s_n)$, the non-zero-sided tensors, which do not have such a side. The origin element is the element of the tensor at location $(1, 1, \ldots, 1)$.*

$$\mathcal{B}(s_1, \ldots, s_n) = \mathcal{C}^{s_1 \times \cdots \times s_n} \tag{1.6}$$

$$\mathcal{Z}(s_1, \ldots, s_n) = \{T \in \mathcal{B}(s_1, \ldots, s_n) \mid \exists i \forall j \forall s_j \; T_{s_1, \ldots, s_{i-1}, 1, s_{i+1}, \ldots, s_n} = 0\} \tag{1.7}$$

$$\bar{\mathcal{Z}}(s_1, \ldots, s_n) = \mathcal{B}(s_1, \ldots, s_n) \setminus \mathcal{Z}(s_1, \ldots, s_n) \tag{1.8}$$

So in 1 dimension, the zero-sided vectors are those whose with a 0 as the first element. In 2 dimensions, the zero-sided matrices are those with a top row of all zeros, or a leftmost column of all zeros, or both.

It is useful to define a matching function H that can be used in the construction of various functions over these tensors. The function returns 1 iff two tensors have elements that match, where the * symbol is treated as matching any color.

Definition 1.10 *Given n-dimensional tensors over $\mathcal{C}_*$, the function $H : \mathcal{C}_*^{s_1 \times \cdots \times s_n} \times \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \{0, 1\}$ is defined as*

$$H(A, B) = \begin{cases} 1 & \text{if } \forall i \forall s_i \; A_{s_1, \ldots, s_n} = B_{s_1, \ldots, s_n} \\ & \quad \vee A_{s_1, \ldots, s_n} = * \\ & \quad \vee B_{s_1, \ldots, s_n} = * \\ 0 & \text{otherwise} \end{cases} \tag{1.9}$$

Given an n-dimensional tensor, it is useful to unwrap it into a 1D string of characters. This will be done in *row major order*. For matrices, this means the elements will be read from left to right across the top row, then left to right across the second row, and so on down to the bottom row. Tensors of other dimensionalities are unwrapped similarly, with the last dimension changing most quickly, and the first dimension changing most slowly. It is useful to have a function $V_{num}(T)$ that unwraps the elements of tensor T, then converts the resulting string to an integer by treating it as a number in base c, with the first element being the most significant digit, and the last being the least significant.

Definition 1.11 *An n-dimensional tensor $\mathcal{A}$ with elements in $\mathcal{C}$ can be converted to an integer by the function $V_{num} : \mathcal{C}^{s_1 \times \cdots \times s_n} \rightarrow \mathbb{N}$, which treats the elements of the tensor as digits base k, where the elements are taken in row major order, treating the first as the least significant digit, and the last as the most significant.*

$$V_{num}(A) = \sum_{i_1=1}^{s_1} \sum_{i_2=1}^{s_2} \cdots \sum_{i_n=1}^{s_n} A_{i_1,i_2,\ldots,i_n} \prod_{j=1}^{n} k^{(i_j-1)\prod_{m=j+1}^{n} s_m} \tag{1.10}$$

For this definition, the rightmost product is understood to be 1 for all cases where the lower bound exceeds the upper.

Definition 1.12 *An n-dimensional tensor with elements in $\mathcal{C}$ can be converted to a binary vector by the function $V_t : \mathcal{C}^{s_1 \times \cdots \times s_n} \rightarrow \{0,1\}^{(k^{s_1 s_2 \cdots s_n})}$, which is defined as*

$$V_t(M) = v \text{ where } v_i = \begin{cases} 1 & \text{if } i = V_{num}(M) + 1 \\ 0 & \text{otherwise} \end{cases} \tag{1.11}$$

The vector $V_t(M)$ has one element for each possible color pattern for a tensor of the same size as M. That vector will be all zeros, except for a 1 in the position corresponding to the pattern M.

Definition 1.13 *A function $f : \mathcal{C}^{s_1 \times \cdots \times s_n} \rightarrow \mathbb{R}$ can be converted to a real vector with $k^{s_1 s_2 \ldots s_n}$ elements by the function $V : (\mathcal{C}^{s_1 \times \cdots \times s_n} \rightarrow \mathbb{R}) \rightarrow \mathbb{R}^{k^{s_1 s_2 \cdots s_n}}$, which is defined as*

$$V(f) = \sum_{M \in \mathcal{B}(s_1,\ldots,s_n)} f(M) \cdot V_t(M) \tag{1.12}$$

This vector is a convenient way to represent an energy function. It completely specifies the energy function, by listing the output of the function for every possible input. We will define various classes of energy functions by simultaneous linear equations, treating the elements of this vector as the variables.

Note that the energy function window is independent of the CA neighborhood. Energy functions can be defined over regions different from the scope of the transition rule of the CA. Our work with 1D CAs in [1], for example, has identified conserved energy functions with windows of size 1×5, 1×6 and larger, for CAs that have neighborhoods of size 1×3.

Definition 1.14 *Given tensor M of size $m_1 \times \cdots \times m_n$, which is a region within an n-dimensional universe, and given an energy window size of $s = (s_1, \ldots, s_n)$, a vector representing the total energy of all energy windows that fit within M can be found with the function*

$$e : \mathbb{N}^n \times \mathcal{C}^{m_1 \times \cdots \times m_n} \rightarrow \mathbb{N}^{k^{s_1 s_2 \cdots s_n}}$$

which is defined as

$$e(s, M) = \sum_{i_1=1}^{m_1-s_1+1} \sum_{i_2=1}^{m_2-s_2+1} \cdots \sum_{i_n=1}^{m_n-s_n+1} V_t(M_{i_1 \ldots i_1+s_1-1,\ldots,i_n \ldots i_n+s_n-1}) \tag{1.13}$$

The $e(s, M)$ function slides the energy window to all possible positions that fit entirely within the matrix M, and finds the energy at each position. It then sums all the energies coming from identical

patterns, and constructs a vector with the total energy derived from each possible pattern. The sum of the elements of this vector would simply be the total energy of M. But it is useful to maintain the vector of separate values when generating sets of linear equations that define the trivials, the nontrivials, or the conserved functions.

Definition 1.15 *For a positive integer n, the function $N : \mathbb{N}^n \to \mathbb{N}$ is defined as*

$$N(s_1, \ldots, s_n) = \sum_{b=1}^{2^n-1} k^{\prod_i s_i - b_i} (-1)^{1+\sum_i b_i} \tag{1.14}$$

where b_i is the ith bit of integer b written in binary, with bit 1 being least significant and bit n being most.

In 1 and 2 dimensions this reduces to:

$$N(c) = k^{c-1} \tag{1.15}$$

$$N(r, c) = k^{(r-1)c} + k^{r(c-1)} - k^{(r-1)(c-1)} \tag{1.16}$$

It will be proved below that this gives the cardinality of many of the sets that will be considered here. It equals the number of zero-sided tensors of a given size, the number of trivials, and the number of unit complements. And when subtracted from a simple power of 2, it yields the number of non-zero-sided tensors, the number of equations defining the conserved functions, and the number of equations defining the nontrivials. These terms are defined and the counts proved below.

Definition 1.16 *In n dimensions, the seven transforms that operate on tensors of size $s_1 \times \cdots \times s_n$*

$$P_C : \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \mathcal{C}_*^{s_1 \times \cdots \times s_n} \tag{1.17}$$
$$P_* : \mathbb{N} \times \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \mathcal{C}_*^{s_1 \times \cdots \times s_n} \tag{1.18}$$
$$P_{rot} : \mathbb{N} \times \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \mathcal{C}_*^{s_1 \times \cdots \times s_n} \tag{1.19}$$
$$P_{LD} : \mathbb{N} \times \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \mathcal{C}_*^{s_1 \times \cdots \times s_n} \tag{1.20}$$
$$P_{RD} : \mathbb{N} \times \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \mathcal{C}_*^{s_1 \times \cdots \times s_n} \tag{1.21}$$
$$P_L : \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \mathcal{C}_*^{s_1 \times \cdots \times s_n} \tag{1.22}$$
$$P_R : \mathcal{C}_*^{s_1 \times \cdots \times s_n} \to \mathcal{C}_*^{s_1 \times \cdots \times s_n} \tag{1.23}$$

are defined to be:

$$P_C(M) = M' \text{ where } M'_{i_1,\ldots,i_n} = \begin{cases} 0 & \text{if } \forall j\ i_j = \lceil s_j/2 \rceil \\ M_{i_1,\ldots,i_n} & \text{otherwise} \end{cases} \tag{1.24}$$

$$P_*(d, M) = M' \text{ where } M'_{i_1,\ldots,i_n} = \begin{cases} * & \text{if } i_d = 1 \\ M_{i_1,\ldots,i_n} & \text{otherwise} \end{cases} \tag{1.25}$$

$$P_{rot}(d, M) = M' \text{ where } M'_{i_1,\ldots,i_n} = M_{i_1,\ldots,i_{d-1},\ 1+(i_d \bmod s_d)\ ,i_{d+1},\ldots,i_n} \tag{1.26}$$

$$P_{LD}(d, M) = \begin{cases} P_*(d, M) & \text{if } \forall j \forall i_j\ M_{i_1,\ldots,i_{d-1},1,i_{d+1},\ldots,i_n} \in \{0, *\} \\ M & \text{otherwise} \end{cases} \tag{1.27}$$

$$P_{RD}(d, M) = \begin{cases} P_{rot}(P_*(d, M)) & \text{if } \forall j \forall i_j\ M_{i_1,\ldots,i_{d-1},1,i_{d+1},\ldots,i_n} \in \{0, *\} \\ M & \text{otherwise} \end{cases} \tag{1.28}$$

$$P_L(M) = P_{LD}(1, P_{LD}(2, \ \ldots\ P_{LD}(n, M)\ \ldots)) \tag{1.29}$$

$$P_R(M) = P_{RD}(1, P_{RD}(2, \ \ldots\ P_{RD}(n, M)\ \ldots)) \tag{1.30}$$

The function $P_{rot}(d, M)$ rotates the elements of tensor M along dimension d, so that one side that included the origin moves to the opposite side. The function P_C sets the central element to zero. The function P_L transforms a zero-sided tensor by replacing the 0 elements on each all-zero side with * elements. And P_R does the same, then rotates it so each modified side moves to the opposite side. The functions P_*, P_{LD}, and P_{RD} are only used here to define the other functions, and won't be used again.

The following gives three examples of P_L and P_R applied to zero-sided matrices of size 3×5. In each example, M is a zero-sided matrix, where the all-zero side is on the left, top, and both, respectively:

$$M = \begin{matrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{matrix} \qquad P_L(M) = \begin{matrix} * & 1 & 1 & 1 & 1 \\ * & 0 & 1 & 0 & 1 \\ * & 0 & 1 & 0 & 0 \end{matrix} \qquad P_R(M) = \begin{matrix} 1 & 1 & 1 & 1 & * \\ 0 & 1 & 0 & 1 & * \\ 0 & 1 & 0 & 0 & * \end{matrix} \tag{1.31}$$

$$M = \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{matrix} \qquad P_L(M) = \begin{matrix} * & * & * & * & * \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{matrix} \qquad P_R(M) = \begin{matrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ * & * & * & * & * \end{matrix} \tag{1.32}$$

$$M = \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{matrix} \qquad P_L(M) = \begin{matrix} * & * & * & * & * \\ * & 1 & 0 & 0 & 0 \\ * & 1 & 0 & 1 & 0 \end{matrix} \qquad P_R(M) = \begin{matrix} 1 & 0 & 0 & 0 & * \\ 1 & 0 & 1 & 0 & * \\ * & * & * & * & * \end{matrix} \tag{1.33}$$

Definition 1.17 *The function $P_Z : C^{s_1 \times \cdots \times s_n} \to C^{(2s_1-1) \times \cdots \times (2s_n-1)}$ takes a small n-dimensional tensor and pads it with zero elements on many of its sides to create a large n-dimensional tensor. In each dimension, if the small tensor was of size s_i in that dimension, then the large tensor will be of size $2s_i - 1$*

in that dimension. The zero elements are added in such a way that the last nonzero element in the original tensor becomes the center element in the new tensor.

For example,

$$P_Z\left(\begin{array}{|ccccc|}\hline 1&0&1&1&0\\ 0&1&0&0&0\\ 0&0&0&0&0\\ \hline\end{array}\right) = \begin{array}{|ccccccccc|}\hline 0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&1&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ \hline\end{array} \tag{1.34}$$

In this 2D example, the small matrix M is of size 3×5, and $P_Z(M)$ is of size $(2\cdot3-1)\times(2\cdot5-1) = 5\times9$. Note that this M happens to have 4 nonzero elements, arranged in a sort of V shape. If the elements of M are read in row major order (i.e. left to right across the top row, then left to right on the second row, etc.), then the last nonzero element to be read is the bottom of the V. The P_Z function pads with zeros in such a way as to yield a large matrix of the correct size, with that last nonzero element in the exact center of the large matrix.

Definition 1.18 *For a given tensor size $s_1 \times \cdots \times s_n$, the set $\mathcal{T}$ is defined to be the following set of functions*

$$\mathcal{T}(s_1, \ldots, s_n) = \{f_M \mid M \in \mathcal{Z}(s_1 \times \cdots \times s_n)\} \tag{1.35}$$

where

$$f_M(x) = \begin{cases} 1 & \text{if } M = \mathbf{0} \\ H(x, P_L(M)) - H(x, P_R(M)) & \text{otherwise} \end{cases} \tag{1.36}$$

2 Theoretical results

Proofs of the theorems below are provided in a separate appendix available from the authors.

Theorem 2.1 *The cardinality of the set $\mathcal{Z}(s_1, \ldots, s_n)$ is $N(s_1, \ldots, s_n)$.*

Theorem 2.2 *The cardinality of the set $\bar{\mathcal{Z}}(s_1, \ldots, s_n)$ is $k^{s_1 s_2 \ldots s_n} - N(s_1, \ldots, s_n)$.*

Theorem 2.3 *The set of coefficient vectors for one minimal set of linear equations that define the trivial conserved functions with energy windows of size $s = (s_1, \ldots, s_n)$ is $\{e(s, P_Z(A)) - e(s, P_C(P_Z(A))) \mid A \in \bar{\mathcal{Z}}(s_1, \ldots, s_n)\}$.*

Theorem 2.4 *The set of coefficient vectors for one set of linear equations that defines the conserved functions with energy windows of size $s = (s_1, \ldots, s_n)$ for cellular automaton rule R with neighborhood of size $w = (w_1, \ldots, w_n)$ is*

$$\{e(s, P_Z(A)) - e(s, P_C(P_Z(A)) - e(s, T(R, P_Z(A))) + e(s, T(R, P_C(P_Z(A))))) \mid A \in \bar{\mathcal{Z}}(s_1 + w_1 - 1, \ldots, s_n + w_n - 1)\}$$

Theorem 2.5 *The set $\mathcal{T}(s_1, \ldots, s_n)$ is a basis set for the space of all trivial additive conserved functions with energy windows of size $s_1 \times \cdots \times s_n$.*

Theorem 2.6 *A complement of the coefficient vectors for the equations defining the trivials for energy windows of size* $s_1\times,\ldots,\times s_n$ *is* $\{V_t(M) \mid M \in \mathcal{Z}\}$.

Note that by the definition of complements, this implies that when searching for conserved functions, without loss of generality we can constrain the energy functions to assign an energy of 0 to any window that is a zero-sided tensor. This corresponds to deleting certain columns in the matrix that defines the conserved functions. After that deletion, there will be solutions to those equations if and only if nontrivials exist. If such solutions do exist, then those solutions are guaranteed to be nontrivial conserved functions, and the union of those solutions with the trivials will span the space of conserved functions. This allows faster searches for nontrivials.

Figure 1 summarizes all the theorems of this paper, giving four examples of the M matrix for each concept. Figure 2 applies the ideas of this paper to the results of [1] and [3], expressing the basis functions as a linear sum of the matching H-functions of Definition 1.10.

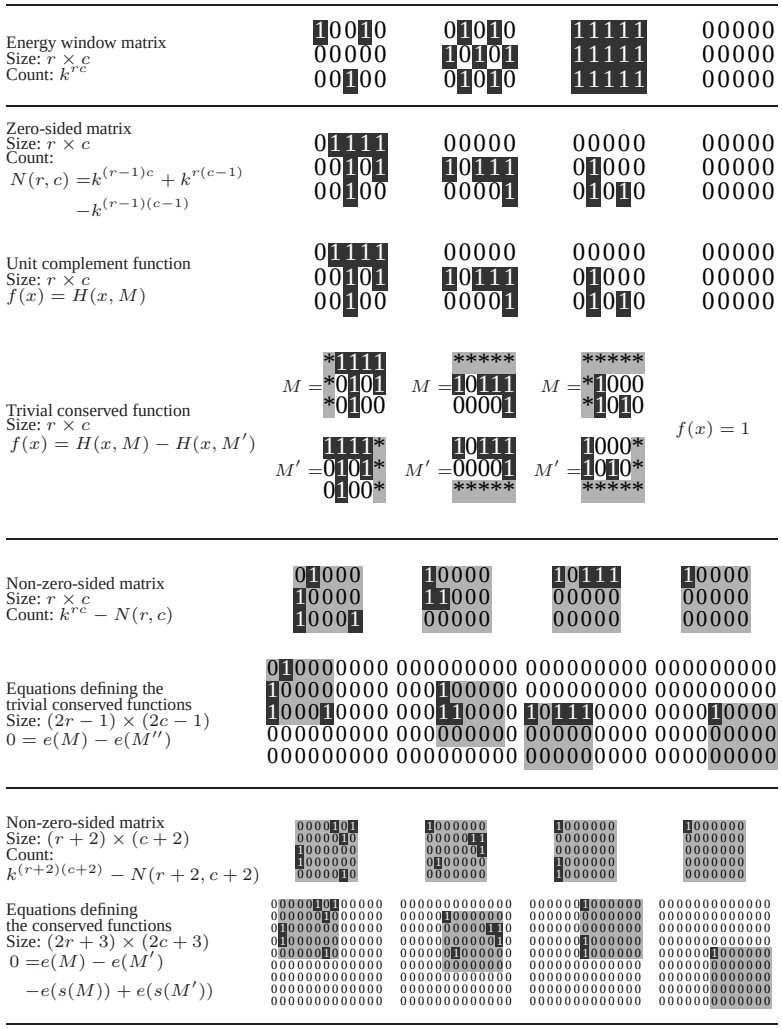


Fig. 1: Summary of the main theoretical results of this paper, with four examples of each concept. The proofs are for arbitrary dimensions, neighborhood sizes, and number of colors, but the figure shows only 2D examples, for a CA with a 3×3 neighborhood, and $k = 2$ colors. In each case, M' is M with the central bit set to 0. For the equations, the large matrix is formed by padding the small matrix with zeros such that the last 1 bit ends up in the center of the large matrix (where "last" is the last 1 found when traversing the elements in row major order). In each of the four sections, the listed concepts all have the same count. For example, the number of zero-sided matrices of a given size equals the number of unit complement functions, which equals the number of trivials.

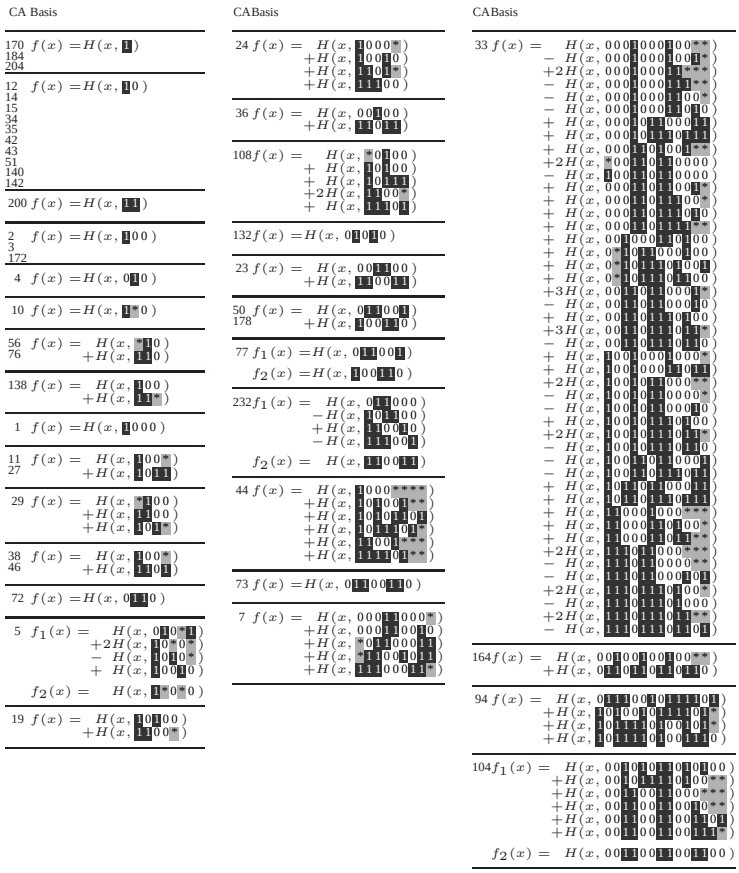


Fig. 2: 1D Basis functions. For each CA, this lists the lowest-order nontrivial conserved functions. The given functions, combined with the trivials, constitute a basis set for the space of all conserved functions for that CA. The table contains all 88 of the non-isomorphic primitive CAs, except those that are known to have no nontrivials (0,8,32,40,128,136,160,168,60,30,90,154), and those that have no known nontrivials and have been proved to have none at least up to and including size 16 energy windows (106,150,6,9,13,18,22,25,26,28,37,41,45,54,57,58,62,74,78,105,110, 122,126,130,134,146,152,156,162).

3 Computational results

The challenge in identifying cellular automata with a nontrivial additive energy conservation function (hereafter referred to as a "nontrivial") is the enumeration of the trivial functions and their elimination from the solution space. The actual calculation of the nontrivials can then be reduced to the calculation of the null space of the system of corresponding state space equations. Thus the theorems and definitions of the previous section may be used as the basis for computational identification of cellular automata with nontrivials of various orders. Computationally, this proceeds as follows:

1) Choose a CA and energy window size (s_1, s_2).

2) For all possible matrices M given by Theorem 2.4, generate the corresponding state space equations.

3) To remove the trivials from the solution space, delete the columns associated with the zero-sided tensors as determined by Theorem 2.6. This has the additional benefit of significantly reducing the size of the energy vectors and, therefore, the state space matrix as a whole.

4) Determine the rank of the resulting matrix. If it is full rank, the system of equations has no solution, and therefore no nontrivial exists for the given CA and window size. If the matrix is rank-deficient, a nontrivial exists. It is completely characterized by the basis vectors that are the columns of the matrix's null space.

In [1], we gave a complete taxonomy of binary nontrivials for 1D cellular automata up for energy windows up to size 16. Using the definitions and theorems previously presented, we now extended these results to binary 2D automata, for energy windows up to size 9.

There are a total of k^{k^9} k-colored 2D cellular automata (ignoring isomorphic entries). This number is so large that any investigation other than a random sampling is effectively impossible. Accordingly, drawing substantive conclusions about unrestricted 2D cellular automata seems to the authors extraordinarily difficult. To reduce the scope of the problem and make a more complete investigation possible, we consider only *outer totalistic* CAs: Those for which the next state of the cell is a function only of the total number of colors of a given type in the region surrounding the cell and the cell itself. For binary CAs, this means that only the total number of 1's in a cell's neighborhood (including its own value) must to be calculated to determine the cell's next state. Conway's Game of Life is a cellular automaton of this type.

Restricting the search space to outer totalistic automata significantly reduces the size of the problem. For a 2D CA, the neighborhood is of size 9, and therefore the total number of occupied cells in a cell's neighborhood ranges from 0 through 8. For binary automata, one of four outcomes are possible: (S)ame, (B)irth, (D)eath, and (F)lip (Flip changes 0 to 1 and vice versa). Thus any outer totalistic CA can be represented as a character string of the form S,B,D,F. Using this notation, if we count the neighbors from 0 to 8 from left to right, Conway's Game of Life would be written as "DDSBDDDDD". We refer to this description at the CA's *rule vector*. Note that the use of symbols S and F permits the incorporation of the central state into the transition rule.

It is known that renumbering the colors of a CA in reverse order and changing the outcomes correspondingly produces an CA identical to the original, up to isomorphism. Using the proposed notation, this corresponds to reversing the order of the letters, swapping S with F, and swapping B with D. The rule vector of every CA can be manipulated in this way to produce a unique and distinct isomorph, so the total number of unique totalistic binary CAs is $4^9/2 = 2^{17}$. This is considerably smaller than the non-totalistic case.

The definitions and theorems in this paper give the dimensions of the matrices to be analyzed as a function of the energy window (independent of the CA being analyzed). We show the matrix sizes for

some 2D examples in Table 1.

Energy window height (s_1)	Energy window width (s_2)	$\lceil log_2\, rows \rceil$	$\lceil log_2\, cols \rceil$
1	2	16	1
1	3	19	2
1	4	23	3
2	2	20	4
1	5	26	4
1	6	29	5
2	3	25	6
1	7	32	6
1	8	35	7
2	4	29	8
1	9	39	8
3	3	30	9

Tab. 1: State matrix sizes for various energy windows

Column three shows the ceiling of the log base 2 of the maximum number of energy vectors needed to determine the existence of a nontrivial. Column four shows the number of entries in each vector. This is given by the total number of possible energy function values ($2^{s_1 s_2}$) minus the number of zero-sided tensors given by Definition 1.15.

Because these matrices have far more rows than columns, we expect almost all of them to be full rank, and therefore few nontrivial conservation functions should exist over the range of cellular automata. Since full rank can be determined very quickly while rank-deficiency cannot be known until all the possible state space vectors given by Theorem 2.3 have been examined for linear independence, it would be inefficient to build the full state space matrix for each CA and then calculate its rank. Instead, we sift the sands of cellular automata through a three-stage computational sieve.

The first stage uses a "quick and dirty" algorithm to discard automata with no nontrivials. This eliminates over 99% of the candidates. The second stage takes automata that have passed the first stage and performs a little more work to try and drive the set of state space matrices to full rank. This eliminates about another 90% of the candidates it analyzes. The third stage operates only on automata that have passed the first two stages, performing exact arithmetic using all the optimizations of Theorem 2.3 to determine whether or not a given CA has a nontrivial conservation function. If it does, its basis is calculated and reported. Each stage is implemented in MATLAB.

In stage I, we compute the energy vector of Definition 1.14 for one tensor at a time, attempting to add it to an existing energy vector set via Gaussian elimination to ensure that the rows in the state space matrix at any time are always linearly independent. Before such addition, however, we delete the columns corresponding to the zero-sided tensors for the indicated energy window. The total number of deleted

columns is given by Definition 1.15. None of the optimizations discussed in the proof of Theorem 2.3 are performed at this stage. Instead, universe states are generated randomly, the energy vectors of their corresponding tensors are calculated, and Gaussian elimination is performed on each vector relative to those energy vectors already admitted into the state space matrix. When the number of linearly independent energy vectors is equal to the number of columns (the number of possible energy function values minus the number of zero-sided tensors), full rank has been achieved, and the CA/energy window pair under test is known not to correspond to a nontrivial conservation function.

Since states are generated randomly in this stage, as opposed to exhaustive enumeration of the appropriate tensors as given by Theorem 2.3, the number of states N to try before giving up on the possibility of reaching full rank is a user-definable parameter. Empirically, we have found that setting N at 32x the maximum rank of the matrix gives a good tradeoff between quick computation on the one hand and admitting too many false positives on the other.

During this stage, all arithmetic is performed modulo a small prime, to eliminate the possibility of roundoff error or overflow. If full rank is reached, the matrix would be full rank in exact arithmetic as well, so the answer is correct. If full rank is not reached within the indicated time window, the matrix may or may not be rank-deficient, so the CA is marked as a candidate for stage II computation.

In stage II, candidate CA/window pairs that pass through the first stage are subject to repeated random state generation with a larger value of N for multiple attempts. No other optimizations are performed at this time. If no full rank matrix is produced (i.e. no linearly independent energy vector set of the cardinality given by Definition 1.14 is found), the pair is marked for analysis by stage III.

Stage III computation employs on-the-fly Gaussian elimination for one-at-a-time energy vector generation, similar to the first two stages, but using double precision arithmetic and enumerating the state space exactly as described in the proof of Theorem 2.3. To keep the computations from overflowing, vectors are reduced modulo the GCD of all their nonzero entries during this process, which means this stage is the most computationally intensive. If Gaussian elimination on the entire set of energy vectors does not produce a linearly independent set of Definition 1.14 cardinality, then constructed state space matrix has a null space. That null space is calculated, and reported as the basis for all nontrivial conservation functions for that particular CA/window combination.

To guard against the possibility of numerical error, the largest value observed during stage III calculation is tracked and reported, to ensure that any possibility of overflow or loss of precision will be detected. For all calculations reported here, this maximum value has always been well below that which could induce error in double precision arithmetic. So we are confident our results are correct. Nonetheless, as an added safety check, we have implemented code which accepts as input a CA, an energy window, and a stage III basis set reported as characterizing a nontrivial. It tests each vector in the basis set over large numbers of randomly selected states by evaluating the energy function through brute force dot product calculation. In all cases, the resulting functions reported by stage III were conserved.

Table 2 shows the results of our computations for all outer totalistic binary 2D cellular automata up to isomorphism, for all energy windows up to order 9. It extends [1] to give a complete taxonomy of conservation functions for all automata of this type. Figures 3 and 4 are similar to Figure 2, extended to two dimensions. Figure 5 summarizes our current knowledge of 1D conservation functions.

CA#	Rule	rule vec (num neighbors)									min NCF	basis size	comments
		0	1	2	3	4	5	6	7	8			
0	S0123456789	S	S	S	S	S	S	S	S	S	1x1	n/a	identity, conserves all
2	S12345678	D	S	S	S	S	S	S	S	S	1x2	1	conserves [11] pairs
8	S02345678	S	D	S	S	S	S	S	S	S	2x2	5	conserves 2x2 patterns with $\geq$ 3 1's
10	S2345678	D	D	S	S	S	S	S	S	S	2x2	5	identical to 8
21	B012/S012345678	B	B	B	S	S	S	S	S	S	3x3	1	
32	S01345678	S	S	D	S	S	S	S	S	S	2x2	1	conserves 2x2 pattern with all 1's
34	S1345678	D	S	D	S	S	S	S	S	S	2x2	1	identical to 32
40	S0345678	S	D	D	S	S	S	S	S	S	2x2	1	identical to 32
42	S345678	D	D	D	S	S	S	S	S	S	2x2	1	identical to 32
16386	B7/S12345678	D	S	S	S	S	S	S	B	S	2x2	4	
16387	B07/S12345678	F	S	S	S	S	S	S	B	S	3x3	11	
21845	B01234567/S8	B	B	B	B	B	B	B	B	S	3x3	1	conserves ring of 1's around a 0
65532	B1234567/S8	S	F	F	F	F	F	F	F	S	2x3	1	
65533	B01234567/S08	B	F	F	F	F	F	F	F	S	2x3	1	identical to 65532
65534	B1234567/S8	D	F	F	F	F	F	F	F	S	2x3	1	identical to 65532
65535	B01234567/S8	F	F	F	F	F	F	F	F	S	2x3	1	identical to 65532
65537	B08/S012345678	B	S	S	S	S	S	S	S	B	2x3	7	
65538	B8/S12345678	D	S	S	S	S	S	S	S	B	2x2	8	
65539	B08/S12345678	F	S	S	S	S	S	S	S	B	2x3	7	identical to 65537
65541	B018/S012345678	B	B	S	S	S	S	S	S	B	3x3	1	conserves [001 011 010]
65545	B08/S234567	B	D	S	S	S	S	S	S	B	2x3	1	conserves the difference between [101 111] and [111 101]
65546	B8/S234567	D	D	S	S	S	S	S	S	B	2x2	4	conserves 2x2 patterns with $\geq$ 3 1's
65547	B08/S2345678	F	D	S	S	S	S	S	S	B	2x3	1	identical to 65545
65549	B018/S02345678	B	F	S	S	S	S	S	S	B	3x3	1	identical to 65541
81921	B078/S012345678	B	S	S	S	S	S	S	B	B	2x3	1	
81923	B078/S12345678	F	S	S	S	S	S	S	B	B	2x3	1	identical to 81921
131069	B012345678/S08	B	F	F	F	F	F	F	F	B	2x3	1	identical to 65532
131070	B12345678/S8	D	F	F	F	F	F	F	F	B	2x3	1	identical to 65532
131071	B012345678/S8	F	F	F	F	F	F	F	F	B	2x3	1	identical to 65532
131073	B0/S01234567	B	S	S	S	S	S	S	S	D	2x3	2	
131075	B0/S1234567	F	S	S	S	S	S	S	S	D	2x3	2	identical to 131073
131077	B01/S01234567	B	B	S	S	S	S	S	S	D	3x3	1	identical to 65541
131081	B0/S0234567	B	D	S	S	S	S	S	S	D	3x3	9	
131083	B0/S234567	F	D	S	S	S	S	S	S	D	3x3	9	identical to 131081
131085	B01/S234567	B	F	S	S	S	S	S	S	D	3x3	1	identical to 65541
147459	B07/S1234567	F	S	S	S	S	S	S	B	D	3x3	9	
163483	B0/S123456	F	S	S	S	S	S	S	D	D	3x3	1	conserves [011 100 101]
180227	B07/S123456	F	S	S	S	S	S	S	F	D	3x3	1	identical to 163843
196605	B01234567/S0	B	F	F	F	F	F	F	F	D	2x2	4	
196607	B01234567	F	F	F	F	F	F	F	F	D	2x2	4	Identical to 196605
196611	B08/S1234567	F	S	S	S	S	S	S	S	F	2x3	2	Identical to 131073
196619	B08/S234567	F	D	S	S	S	S	S	S	F	3x3	9	Identical to 131081
262143	B012345678	F	F	F	F	F	F	F	F	F	1x2	1	Conserves [10] pairs

Tab. 2: Conservation functions of order ≤ 9 for 2D CA's

The first three columns of Table 2 are all different ways of identifying the same automaton. The first column is the decimal integer represented by a CA's rule vector, obtained by treating the symbols S,B,D,F as the integers 0,1,2,3 respectively, and viewing the rule vector as a number in base 4 with the most significant digit on the right. The second column shows the CA rule using the notation in [8]. Column three is the CA's rule vector.

Columns four through six describe the nontrivial conservation function found. Column four shows the dimensions of the energy window at which the first nontrivial was discovered. Column five shows the number of basis vectors in the null space of the CA's state matrix for an energy window of the indicated size. Column six contains, where appropriate, comments describing the conservation function. A blank entry in this column means that either no simple description exists or that describing the pattern would be too complex to fit within the indicated space.

Symmetry arguments will show that analogous conservation functions for any $m \times n$ window can also be found for one that is $n \times m$. Thus the only energy windows examined were those that were at least as wide as they were tall.

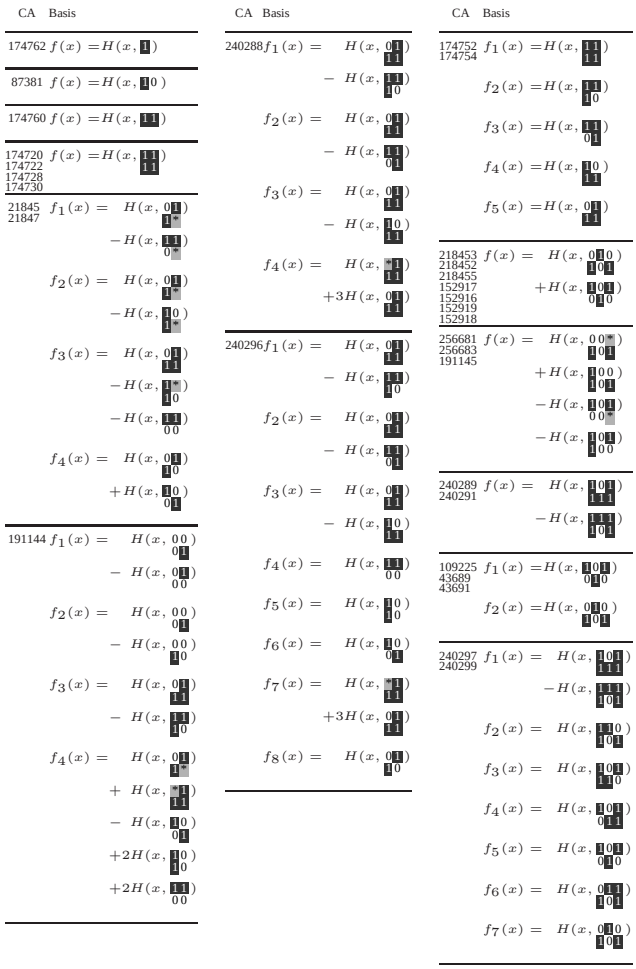


Fig. 3: 2D Basis functions. For each CA, this lists the lowest-order nontrivial conserved functions. The given functions, combined with the trivials, constitute a basis set for the space of all conserved functions for that CA. The table contains all of the non-isomorphic, 2-color, 3×3 neighborhood, outer totalistic CAs that have nontrivials of size 2×3 or smaller (the 3×3 nontrivials are shown in Figure 4).

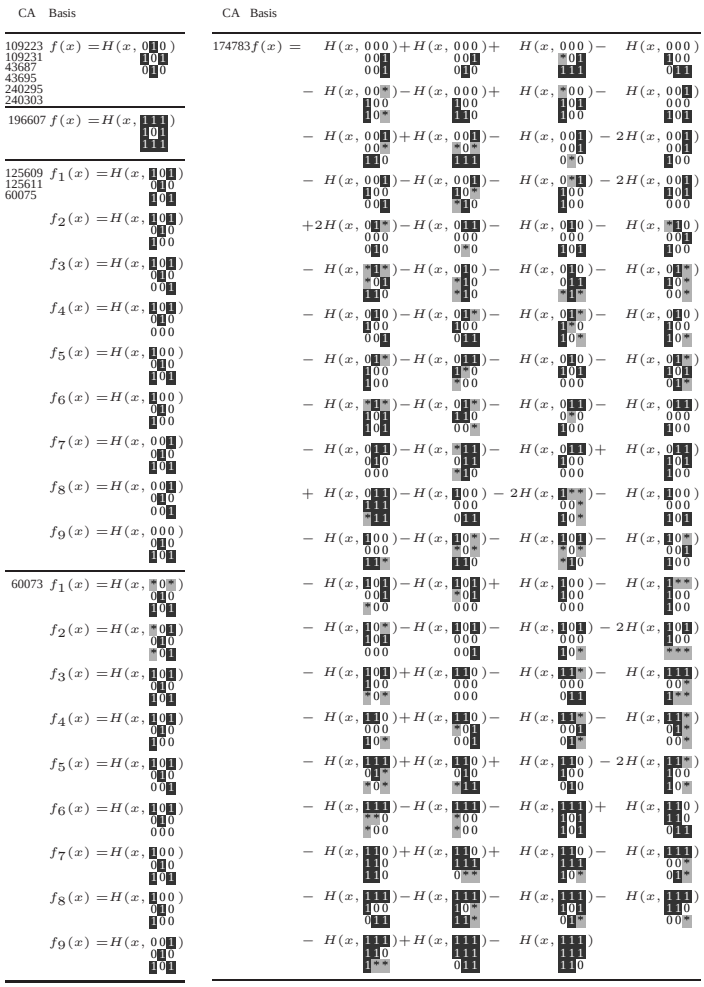
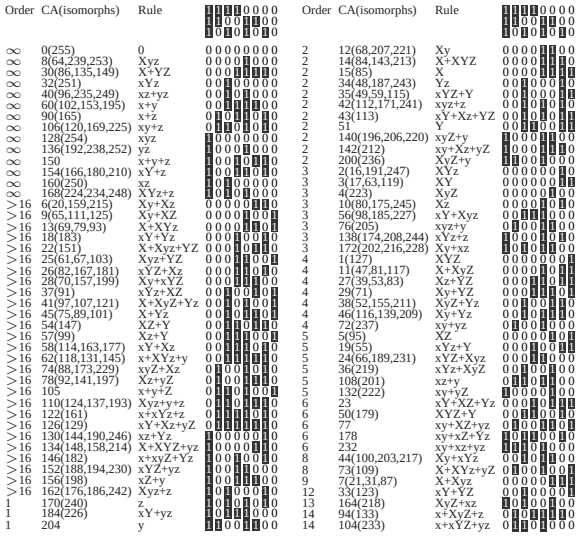


Fig. 4: 2D Basis functions (continued). These are the 3×3 nontrivials, continued from figure 3.



Order	CA(isomorphs)	Rule	Order	CA(isomorphs)	Rule
∞	0(255)	0	2	12(68,207,221)	Xy
∞	8(64,239,253)	Xyz	2	14(84,143,213)	X+XYZ
∞	30(86,135,149)	X+YZ	2	15(85)	X
∞	32(251)	xYz	2	34(48,187,243)	Yz
∞	40(96,235,249)	xz+yz	2	35(49,59,115)	xYZ+Y
∞	60(102,153,195)	x+y	2	42(112,171,241)	xyz+z
∞	90(165)	x+z	2	43(113)	xY+Xz+YZ
∞	106(120,169,225)	xy+z	2	51	Y
∞	128(254)	xyz	2	140(196,206,220)	xyZ+y
∞	136(192,238,252)	yz	2	142(212)	xy+Xz+yZ
∞	150	x+y+z	2	200(236)	XyZ+y
∞	154(166,180,210)	xY+z	3	2(16,191,247)	XYz
∞	160(250)	xz	3	3(17,63,119)	XY
∞	168(224,234,248)	XYz+z	3	4(223)	XyZ
>16	6(20,159,215)	Xy+Xz	3	10(80,175,245)	Xz
>16	9(65,111,125)	Xy+XZ	3	56(98,185,227)	xY+Xyz
>16	13(69,79,93)	X+XYz	3	76(205)	xyz+y
>16	18(183)	xY+Yz	3	138(174,208,244)	xYz+z
>16	22(151)	X+Xyz+YZ	3	172(202,216,228)	Xy+xz
>16	25(61,67,103)	Xyz+YZ	4	1(127)	XYZ
>16	26(82,167,181)	xYZ+Xz	4	11(47,81,117)	X+XyZ
>16	28(70,157,199)	Xy+xYZ	4	27(39,53,83)	Xz+YZ
>16	37(91)	xYz+XZ	4	29(71)	Xy+YZ
>16	41(97,107,121)	X+XyZ+Yz	4	38(52,155,211)	XyZ+Yz
>16	45(75,89,101)	X+Yz	4	46(116,139,209)	Xy+Yz
>16	54(147)	XZ+Y	4	72(237)	xy+yz
>16	57(99)	Xz+Y	5	5(95)	XZ
>16	58(114,163,177)	xY+Xz	5	19(55)	xYz+Y
>16	62(118,131,145)	x+XYz+y	5	24(66,189,231)	xYZ+Xyz
>16	74(88,173,229)	xyZ+Xz	5	36(219)	xYz+XyZ
>16	78(92,141,197)	Xz+yZ	5	108(201)	xz+y
>16	105	x+y+Z	5	132(222)	xy+yZ
>16	110(124,137,193)	Xyz+y+z	6	23	xY+XZ+Yz
>16	122(161)	x+xYz+z	6	50(179)	XYZ+Y
>16	126(129)	xY+Xz+yZ	6	77	xy+XZ+yz
>16	130(144,190,246)	xz+Yz	6	178	xy+xZ+Yz
>16	134(148,158,214)	X+XYZ+yz	6	232	xy+xz+yz
>16	146(182)	x+xyZ+Yz	8	44(100,203,217)	Xy+xYz
>16	152(188,194,230)	xYZ+yz	8	73(109)	X+XYz+yZ
>16	156(198)	xZ+y	9	7(21,31,87)	X+Xyz
>16	162(176,186,242)	Xyz+z	12	33(123)	xY+YZ
1	170(240)	z	13	164(218)	XyZ+xz
1	184(226)	xY+yz	14	94(133)	x+XyZ+z
1	204	y	14	104(233)	x+xYZ+yz

Fig. 5: Summary of results for the primitive CAs (1D, 2-color, neighborhood of 3 cells). In each half of the table, the first column gives the energy window size for the smallest nontrivial. A value of ∞ indicates that it is known no nontrivial can exist. A value of > 16 indicates that no nontrivial exists with energy window of size 16 or below. The next column has the CA name, and the names of the isomorphic CAs. The next is the formula for the successor function, where cells have state 0 or 1, three consecutive cells are called x, y, z (with capitalized inverses, so X=1-x etc.), and the formula modulo 2 gives the new state for y. Finally, the successor function is shown graphically, giving the new state as a function of the state in that cell and its immediate neighbors (shown at the top of the column).

4 Analysis

Some patterns are clearly visible in Table 2, Figure 3, Figure 4 and Figure 5. For all CA's for which nontrivial conservation functions exist, there is a great deal of homogeneity in the middle range of neighbor counts. For example, any given CA in the table has the same transition rules for neighbor counts 3-6, and most have identical transition rules for neighbor counts 2-7. We conjecture this is combinatorically driven. That is, for the middle range of neighbor counts, there are so many different ways to distribute a fixed number of neighbors among eight cells that a low-order conservation function cannot incorporate them all. By contrast, there is only one way to arrange zero or eight neighbors around a cell, eight ways to arrange one or seven, and so forth. Near the minimum and maximum of the neighbor count range, the number of possible configurations is sufficiently small that a low-order conservation function is more likely to emerge.

We also note that all CA's with rule vectors of the form xFFFFFFFx, xSSSSSSSB, and xDSSSSSSB have nontrivial conservation functions. All CA's of the form xSSSSSSSx have a nontrivial as well, unless exactly one of the x's is 'S'.

Finally, our results show that all known nontrivials correspond to energy windows for which the width and the height differ by no more than one. Whether this holds true for all nontrivials remains an open question.

5 The Game of Life

Because of the special significance of Conway's Game of Life (CA #174666, rule B3/S23, rule vector DDSBDDDDD), we have examined it for nontrivial energy conservation functions up to order 13. None have been found.

6 Conclusions and Future Work

Table 2 and Figures 3 through 5 represent a complete taxonomy of all known nontrivial conservation functions for 1- and 2-dimensional binary cellular automata up to isomorphism. We have discussed some of the patterns we have observed.

[1] introduced the notion of core nontrivials, recognizing that cellular automata could exhibit different nontrivials of higher orders that are not simple extensions of lower ones. We have yet to apply this idea to the automata shown here. Thus the functions we report are only the first core nontrivials found. The existence of multiple cores for 2D binary cellular automata remains an open question. Detecting such cores requires only well-understood modifications to our existing code, and is on our list of future enhancements.

Number-conserving 1D cellular automata [2] are automata with transition rules that conserve the sum of the number of states in a neighborhood. A number-conserving function is one kind of energy conservation function defined in Definition 1.8, where the function is simply the sum of all terms in the window. Our work therefore includes number-conservation as a special case. The theory described here applies to all cellular automata with finite states and arbitrary dimensionality. The results for 2D automata are all new.

Continuing improvements in computing power and further refinements of our codes should enable us to identify nontrivials at increasingly higher orders. The existence of nontrivialss for $m \times n$ energy windows with $|m - n| > 1$ remains an open question. Higher dimensional CAs, non-totalistic CAs, and k-colored CAs could also be explored.

As yet, an elegant, unifying description of cellular automata relating their decision rules and a given energy window to a nontrivial conservation function remains elusive. While the general problem is undecidable, we have mapped out the space for lower orders and binary outer totalistic CAs well enough to suggest some ideas for a more elegant classification scheme than the present ad hoc one we are currently forced to adopt. Such a scheme may in fact exist, or it may remain forever elusive, an fundamentally complex property inherent in the nature of computational automata. We hope further work may yet resolve this question.

7 Errata and Acknowledgments

Readers unfamiliar with automata conservation functions may wish to review [1]. In the course of preparing this paper, we noticed errors in the first three tables of our previous results. For the sake of completeness, we present the necessary corrections to [1] here:

TABLE 1: Replace 98 with 94, replace 40 with 46
TABLE 2: Replace 136 with 200
TABLE 3: Replace 136 with 200, replace 248 with 232

The authors are grateful for the support of the Air Force Academy Center for Cyberspace Research, and to the reviewers for their helpful comments.

References

[1] L. Baird and B. Fagin, *Conservation functions for 1-d automata: Efficient algorithms, new results, and a partial taxonomy*, Journal of Cellular Automata **3** (2008), no. 4, 271–288.

[2] Nino Boccara and Henryk Fuks, *Number-conserving cellular automaton rules*, Fundam. Inform. **52** (2002), no. 1-3, 1–13.

[3] B. Fagin and L. Baird, *New higher-order conservation functions for 1-d cellular automata*, Proceedings of the IEEE Symposium on Artificial Life, April 1-5 2007.

[4] H. Fuks, *Remarks on the critical behavior of second order additive invariants in elementary cellular automata*, Fundamenta Informaticae **78** (2007), 329–341.

[5] T. Hattori and S Takesue, *Additive conserved quantities in discrete-time lattice dynamical systems*, Physica D **49** (1991), 295–322.

[6] L. Kotze and W.H. Steeb, *Finite dimensional integrable nonlinear dynamical systems*, pp. 333–346, World Scientific Publishing, New Jersey, 1998.

[7] M. Pivato, *Conservation laws in cellular automata*, Nonlinearity **15** (2002), 1781–1793.

[8] Wikipedia, *Life-like cellular automaton*, `http://en.wikipedia.org/wiki/Life-like_cellular_automaton`, March 2010.

[9] S. Wolfram, *A new kind of science*, Wolfram Media Inc., 2002.

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 21–30

60/102 Null Boundary Cellular Automata based expander graphs

Sung-Jin Cho[1†] and Un-Sook Choi[2] and Han-Doo Kim[3] and Yoon-Hee Hwang[1] and Jin-Gyoung Kim[1]

[1]*Department of Applied Mathematics, Pukyong National University, Busan 608-737, Korea*
[2]*Department of Media Engineering, Tongmyong University, Busan 626-847, Korea*
[3]*School of Computer Aided Science, Institute of Basic Science, Inje University, KimHae 621-749, Korea*

Expander graphs are useful in the design and analysis of communication networks. Mukhopadhyay et al. introduced a method to generate a family of expander graphs based on nongroup two predecessor single attractor Cellular Automata(CA). In this paper we propose a method to generate a family of expander graphs based on 60/102 Null Boundary CA(NBCA) which is a group CA. The spectral gap generated by our method is maximal. Moreover, the spectral gap is larger than that of Mukhopadhyay et al.

Keywords: Expander graphs, $60/102$ NBCA, Spectral gaps, Bipartite graphs, Eigenvalue.

1 Introduction

Expander graphs were first defined by Bassalygo and Pinsker and their existence first proved by Pinsker in the early 1970s (10). Also expander graphs have utility in computational settings such as in the theory of error correcting codes and the theory of pseudorandomness as well as a tool for proving results in number theory and computational complexity (6; 8; 11). Expander graphs are useful in the design and analysis of communication networks. Mukhopadhyay et al. introduced a method to generate a family of expander graphs based on nongroup two predecessor single attractor Cellular Automata(CA). In this paper we propose a method to generate a family of expander graphs based on 60/102 Null Boundary CA(NBCA) which is a group CA. The merit of our method is that it use regular, modular and cascadable structure of $60/102$ NBCA (1; 3; 4) to generate regular graphs of good expansion property with less storage. The spectral gap generated by our method is maximal. Moreover, the spectral gap is larger than that of Mukhopadhyay et al. (9).

†This work was supported by the Pukyong University Research Fund in 2009(PK-2009-26) .

2 Preliminaries

CA consist of a number of interconnected cells arranged spatially in a regular manner, where the state transition of each cell depends on the states of its neighbors. The CA structure investigated by Wolfram (12) can be viewed as a discrete lattice of sites (cells), where each cell can assume the value either 0 or 1. The next-state of a cell is assumed to depend on itself and on its two neighbors (3-neighborhood dependency). If the next-state function of a cell is expressed in the form of a truth table, then the decimal equivalent of the output is conventionally called the rule number for the cell.

Neighborhood state	111	110	101	100	011	010	001	000	
Next state	0	0	1	1	1	1	0	0	rule 60
Next state	0	1	1	0	0	1	1	0	rule 102

The top row gives all eight possible states of the three neighboring cells (the left neighbor of the ith cell, the ith cell itself, and its right neighbor) at the time of instant t. The second and third rows give the corresponding states of the ith cell at the time of instant $t+1$ for two illustrative CA rules.

Informally, expander graph is a graph $G = (V, E)$ in which every subset S of vertices *expands* quickly, in the sense that it is connected to many vertices in the set $\overline{S}$ of complementary vertices.

Definition 2.1. (8) Suppose $G = (V, E)$ has n vertices. For a subset S of V define the *edge boundary* of S, ∂S, to be the set of edges connecting S to its complement $\overline{S}$. That is, ∂S consists of all those edges (v, w) such that $v \in S$ and $w \notin S$. The *expansion parameter for* G is defined by

$$h(G) \equiv \min_{S:|S|\leq n/2} \frac{|\partial S|}{|S|}$$

where $|X|$ denotes the size of a set X.

Example 2.2. Suppose G is the complete graph with n vertices, i.e., the graph in which every vertex is connected to every other vertex. Then for any vertex in S, each vertex in S is connected to all the vertices in $\overline{S}$, and thus $|\partial S| = |S| \times |\overline{S}| = |S|(n - |S|)$. It follows that the expansion parameter for G is given by

$$h(G) \equiv \min_{S:|S|\leq n/2} (n - |S|) = \lceil \frac{n}{2} \rceil$$

It is a marvellous fact that properties of the *eigenvalue spectrum* of the adjacency matrix $A(G)$ can be used to understand properties of the graph G. This occurs so frequently that we refer to the spectrum of $A(G)$ as *the spectrum of the graph* G. It is useful because the eigenvalue spectrum can be computed quickly, and certain properties, such as the largest and smallest eigenvalue, the determinant and trace, can be computed extremely quickly (8).

Let $G = (V, E)$ be an undirected graph and $A(G)$ be the adjacency matrix of the graph G. And let $\lambda_i(A(G))(1 \leq i \leq n)$ be eigenvalues of $A(G)$. Then $A(G)$ is a real symmetric matrix and thus diagonalized. Without loss of generality we can assume that $\lambda_1(A(G)) \geq \lambda_2(A(G)) \geq \cdots \geq \lambda_n(A(G))$.

Lemma 2.3. (1) Let $\mathbb{C}$ be a CA where state transition matrix T and $\mathbb{C}'$ be the complemented CA derived from $\mathbb{C}$ where state transition operator $\overline{T}$. And let $\overline{T}^p$ denote p times application of the complemented CA operator $\overline{T}$. Then

$$\overline{T}^p f(x) = [I \oplus T \oplus T^2 \oplus \cdots \oplus T^{p-1}]F(x) \oplus T^p f(x)$$

where T is the characteristic matrix of the corresponding noncomplemented rule vector and $F(x)$ is an n-dimensional vector (n=number of cells) responsible for inversion after XNORing. $F(x)$ has '1' entries (i.e., nonzero entries) for CA cell positions where XNOR function is employed and $f(x)$ is the current state assignment of the cells.

3 Properties of the eigenvalue spectrum

In this section, we give properties of the eigenvalue spectrum of the adjacency matrix $A(G)$ of an undirected graph G.

The following three theorems are well-known.

Theorem 3.1. Let G be an undirected d-regular graph whose adjacency matrix is $A(G)$. Then $\lambda_1(A(G)) = d$.

Theorem 3.2. Let G be an undirected d-regular graph. Then G is connected if and only if $\lambda_1(A(G)) > \lambda_2(A(G))$.

Theorem 3.3. Let G be an undirected d-regular graph. Then G is bipartite if and only if $\lambda_i(A(G)) = -\lambda_{n+1-i}(A(G)), i = 1, 2, \cdots, n$.

Now we define the *gap for the d-regular graph G* to be the difference $\Delta(G) \equiv d - \lambda_2(A(G))$.

Theorem 3.4. (2) Let G be a d-regular graph with spectrum $\lambda_1(A(G)) \geq \lambda_2(A(G)) \geq \cdots \geq \lambda_n(A(G))$. Then

$$\frac{\Delta(G)}{2} \leq h(G) \leq \sqrt{2d\Delta(G)}$$

Example 3.5. Let G be an undirected graph with the adjacency matrix $A(G)$ as the following:

$$T = \begin{pmatrix}
0&1&0&0&0&0&0&0&1&2&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&1\\
0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&2&0\\
0&0&0&0&1&0&1&0&0&1&0&0&0&1&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0&0&2&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&1\\
0&0&0&1&0&1&0&0&0&0&2&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
2&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&1&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&2&0&0&1&0&0&0&0&0&0&1\\
0&1&0&0&0&1&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&2&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&1&0&0&0&1&0&0&0&0&0&1&0&1&0\\
0&0&2&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&1&0&0&0&1&0&0&1&0&1&0&0&0&0&0&0
\end{pmatrix}$$

Then $\lambda_1(A(G)) = 4, \lambda_2(A(G)) = \lambda_3(A(G)) = 2\sqrt{2}, \lambda_4(A(G)) = \lambda_5(A(G)) = 2, \lambda_6(A(G)) = \cdots = \lambda_{11}(A(G)) = 0, \lambda_{12}(A(G)) = \lambda_{13}(A(G)) = -2, \lambda_{14}(A(G)) = \lambda_{15}(A(G)) = -2\sqrt{2}, \lambda_{16}(A(G)) = -4$. Moreover, $\Delta(G) = 4 - 2\sqrt{2}$. Thus $2 - \sqrt{2} \leq h(G) \leq 4\sqrt{2 - \sqrt{2}}$.

Since $\lambda_1(A(G)) > \lambda_2(A(G))$ and $\lambda_i(A(G)) = -\lambda_{17-i}(A(G))(i = 1, 2, \cdots, 16)$, G is connected and bipartite.

4 60/102 NBCA based expander graphs

In this section we show a construction of a family of random d-regular graphs using $60/102$ NBCA. Let $\mathbb{C}$ be the n-cell $60/102$ NBCA whose state transition matrix T is as the following:

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

Hereafter we write T by $T =< 60, 102, 102, \cdots, 102 >$.

Clearly the characteristic (resp. minimal) polynomial $c(x)$ (resp. $m(x)$) of T is $c(x) = (x + 1)^n$ (resp. $m(x) = (x + 1)^{n-1}$). Since $m(x) = (x + 1)^{n-1}$, we can obtain the following result. The proof of Theorem 4.1 is very similar to the proof of Theorem 3.4 in (3).

Theorem 4.1. Let $\mathbb{C}$ be the n-cell $60/102$ NBCA with state transition matrix $T =< 60, 102, 102, \cdots, 102 >$. Let $\mathbb{C}'$ be the complemented CA derived from $\mathbb{C}$ with complement vector $(a_1, \cdots, a_{n-1}, 1)^t (a_i \in \{0, 1\}, i = 1, 2, \cdots, n - 1$ where $\mathbf{x^t}$ is the transpose of the given vector $\mathbf{x})$ and state transition operator $\overline{T}$. If $ord(T) = 2^a$, then the following hold:

(a) all the lengths of cycles in $\mathbb{C}'$ are the same.

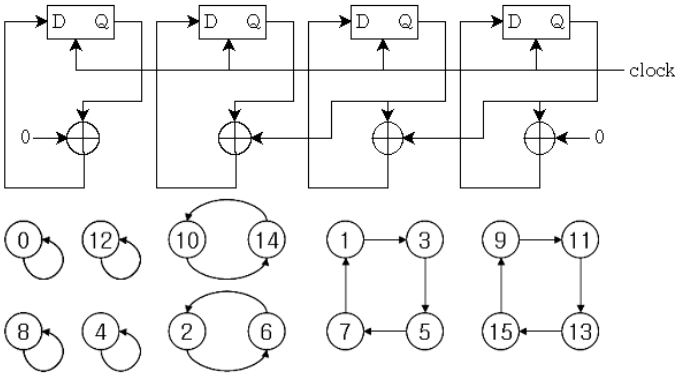
(b) $ord(\overline{T}) = \begin{cases} 2^a, & \text{if } 2^{a-1} < n - 1 < 2^a, \\ 2^{a+1}, & \text{if } n - 1 = 2^{a+1}. \end{cases}$

Remark A By Theorem 4.1, the state transition diagram of $\mathbb{C}'$ does not have any attractor.

Example 4.2. Let $\mathbb{C}$ be the 4-cell $60/102$ NBCA whose state transition matrix is $T =< 60, 102, 102, 102 >$. Then the structure and the state transition diagram of $\mathbb{C}$ are as the following.

Let $F_1 = (0, 0, 0, 1)^t$. Then by Lemma 2.3 $\overline{T}0 = 1$, $\overline{T}1 = 2$, $\overline{T}2 = 7$, $\overline{T}3 = 4$, $\overline{T}4 = 5$, $\cdots$, $\overline{T}14 = 11$ and $\overline{T}15 = 8$. Thus we obtain the state transition diagram G_1 of the state transition operator $\overline{T}$ of the complemented CA $\mathbb{C}'_1$ with complement vector $F_1 = (0, 0, 0, 1)^t$ of $\mathbb{C}$. Also we see that $ord(\overline{T}) = ord(T) = 4$ and all lengths of cycles in $\mathbb{C}$ are all the same by Theorem 4.1. Diagrma

Fig. 2 shows the state transition diagram G_1 and G_2 of the complemented CA with $F_1 = (0, 0, 0, 1)^t$ and $F_2 = (1, 1, 1, 1)^t$ respectively whose two adjacency 8×8 matrices $A(G_1)$ and $A(G_2)$ respectively using Example 4.2 are as the following.



$$T=\begin{pmatrix}1&0&0&0\\0&1&1&0\\0&0&1&1\\0&0&0&1\end{pmatrix}$$

Fig. 1: The structure and the state transition diagram of $\mathbb{C}$

$$A(G_1)=\begin{pmatrix}
0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0
\end{pmatrix}
\quad
A(G_2)=\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0
\end{pmatrix}$$

Let G be the graph obtained by the union of the graphs G_1 and G_2. Then $A(G)$ is as the following:

$$A(G)=\begin{pmatrix}
0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&1\\
1&0&1&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&1&0&0&0&0&0&0&1&0&1&0&1&0&0&0\\
0&0&0&0&1&0&1&0&0&0&1&0&1&0&0&0\\
0&0&0&1&0&1&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&1&0&1&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&1&0&0&0&0&0&0&0&1&0&1\\
1&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&1\\
0&0&1&0&1&0&0&0&1&0&1&0&0&0&0&0\\
0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&1\\
0&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0\\
0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&0\\
1&0&0&0&0&0&0&1&0&0&0&0&0&1&0&1\\
0&0&0&0&0&1&0&1&0&1&0&0&0&0&1&0\\
1&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0
\end{pmatrix}$$

The characteristic polynomial of $A(G)$ is $x^6(x-4)(x+4)(x-2)^4(x+2)^4$. Hence the eigenvalues of $A(G)$ are $\lambda_1 = 4$, $\lambda_2 = \cdots = \lambda_5 = 2$, $\lambda_6 = \cdots = \lambda_{11} = 0$, $\lambda_{12} = \cdots = \lambda_{15} = -2$, $\lambda_{16} = -4$. Therefore by Theorem 3.2 and Theorem 3.3 G is connected and bipartite. Fig. 3 shows the graph G with the adjacency matrix $A(G)$.

Theorem 4.3. Let $\mathbb{C}$ be the 60/102 NBCA whose state transition matrix is T. Let $\mathbb{C}'_1$ (resp. $\mathbb{C}'_2$) be the complemented CA derived from $\mathbb{C}$ with the complement vector $F_1 = (0, *, \cdots, *, 1)^t$ (resp. $F_2 = (1, *, \cdots, *, 1)^t$). Also let $\overline{T}_1 X = TX \oplus F_1$ and $\overline{T}_2 X = TX \oplus F_2$. Let G_1 (resp. G_2) be the graph obtained from $\mathbb{C}'_1$ (resp. $\mathbb{C}'_2$). And let G be the union of two graphs G_1 and G_2 whose adjacency matrix is $A(G_1)$ and $A(G_2)$ respectively. Then G is a bipartite 4-regular graph.

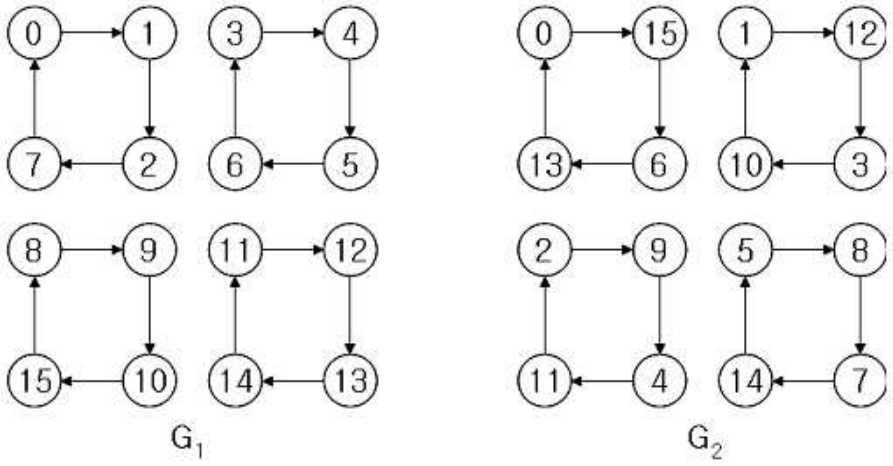


Fig. 2: The state transition diagram G_1 (resp. G_2) of the complemented CA with $F_1 = (0,0,0,1)^t$ (resp. $F_2 = (1,1,1,1)^t$)

Table 1 shows the eigenvalue spectrum of $A(G)$ which is the union of G_1 and G_2. In Table 1 let $F_1 = (0,1,1,1)^t$ and $F_2 = (1,1,0,1)^t$. Then the eigenvalue spectrum of $A(G)$ is $\lambda_1 = 4, \lambda_2 = \cdots = \lambda_5 = 2, \lambda_6 = \cdots = \lambda_{11} = 0, \lambda_{12} = \cdots = \lambda_{15} = -2, \lambda_{16} = -4$. Therefore in this case the graph G is a bipartite 4-regular graph.

Table 2 shows the result of an experimentation performed with the $60/102$ NBCA based regular graph. It measures the value of the two largest eigenvalues for random $60/102$ NBCA based graphs for degree $4, 8, 12$ and 16. Our results show that the spectral gap and hence the expansion increases proportionately with the number of union operations (t). Table 3 shows that the spectral gap by the our method is larger than the spectral gap by Mukhopadhyay's method (9).

Theorem 4.4. Let $\mathbb{C}$ be the n-cell $60/102$ NBCA. Also let $\mathbf{x} = (x_1, x_2, \cdots, x_n)^t$ be a state of the state transition diagram of the state transition matrix T of $\mathbb{C}$. Then the immediate predecessor $\mathbf{y} = (y_1, y_2, \cdots, y_n)^t$ of $\mathbf{x}$ satisfies the following:

$$y_1 = x_1, y_n = x_n, y_k = x_k \oplus y_{k+1} \ \ (k = 2, \cdots, n-1)$$

Remark B It is easy to see that the inverse matrix T^{-1} of T is of the following form.

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 0 & 0 & 1 & \cdots & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

So the required time to get the immediate predecessors is $O(n)$. For the given n-cell 60/102 NBCA, the construction of d-regular graphs which have the maximum spectral gaps depend on the relationship between F_1 and F_2. For example, in Table 1 let $F_1 = (0,0,0,1)^t$ and $F_2 = (1,1,1,1)^t$. Then the spectral

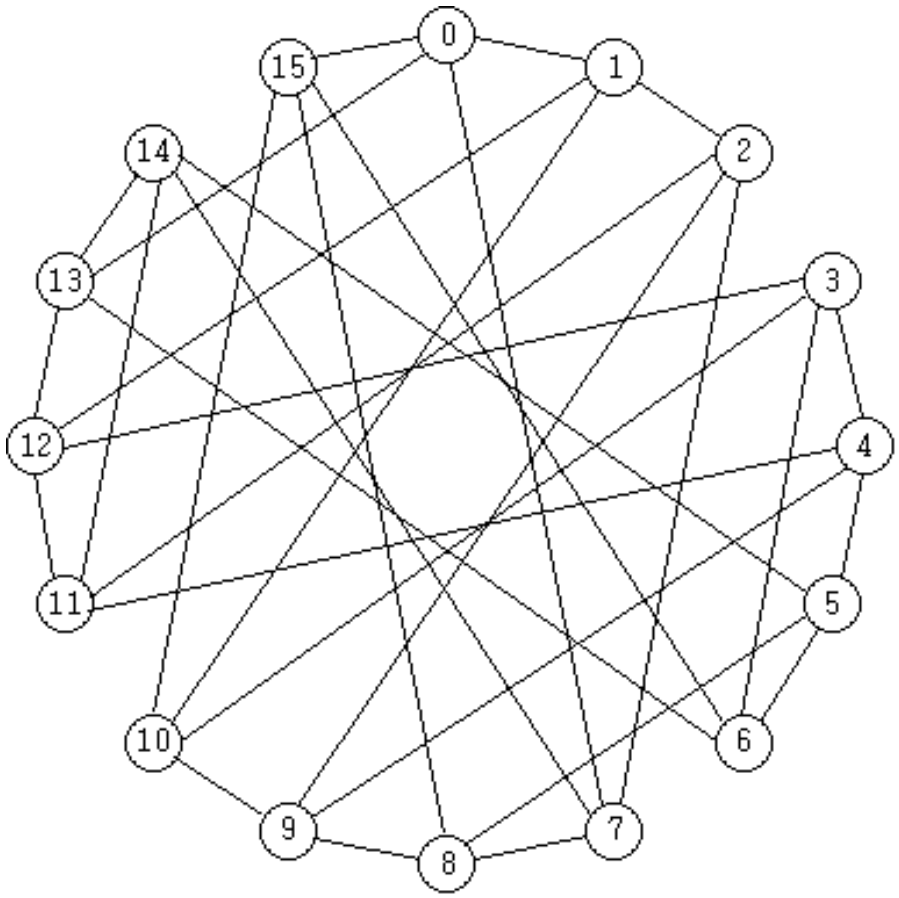


Fig. 3: The graph G

gap is 2 which is the maximum value in the 4-regular graph.

Now let

$$F_{11} = \{(0, a_2, a_3, \cdots, a_{n-2}, 0, 1) | a_i \in \{0, 1\}, i = 2, \cdots, n-2\}$$
$$F_{12} = \{(0, a_2, a_3, \cdots, a_{n-2}, 1, 1) | a_i \in \{0, 1\}, i = 2, \cdots, n-2\}$$
$$F_{21} = \{(1, a_2, a_3, \cdots, a_{n-2}, 1, 1) | a_i \in \{0, 1\}, i = 2, \cdots, n-2\}$$
$$F_{22} = \{(1, a_2, a_3, \cdots, a_{n-2}, 0, 1) | a_i \in \{0, 1\}, i = 2, \cdots, n-2\}$$

and let $U = (F_{11} \times F_{21}) \cup (F_{12} \times F_{22})$.

Choose the complement vectors F_1, F_2 such that $(F_1, F_2) \in U$. Let G_1 (resp. G_2) be the graph with F_1 (resp. F_2). Then we can construct an expander graph where spectral gap is maximal.

Table 1. The eigenvalue spectrum of $A(G)$
The eight vectors on the first row(resp. column) are the complement vectors F_1(resp. F_2)

	0000	0010	0100	0110	0001	0011	0101	0111
1000	-4(2)	-4(1)	-4(2)	-4(1)				
1100	0(10)	-2(4)	0(10)	-2(4)	-4(1)	-4(1)	-4(1)	-4(1)
	4(4)	0(4)	4(4)	0(4)	-2.8284(2)	-2.8284(2)	-2.8284(2)	-2.8284(2)
		2(4)		2(4)	-2(2)	-2(2)	-2(2)	-2(2)
		4(3)		4(3)	0(6)	0(6)	0(6)	0(6)
1010	-4(1)	-4(2)	-4(1)	-4(2)	2(2)	2(2)	2(2)	2(2)
1110	-2(4)	0(10)	-2(4)	0(10)	2.8284(2)	2.8284(2)	2.8284(2)	2.8284(2)
	0(4)	4(4)	0(4)	4(4)	4(1)	4(1)	4(1)	4(1)
	2(4)		2(4)					
	4(3)		4(3)					
1001					-4(2)	-4(1)	-4(2)	-4(1)
1101					0(12)	-2(4)	0(12)	-2(4)
	-2.8284(2)	-2.8284(2)	-2.8284(2)	-2.8284(2)	4(2)	0(6)	4(2)	0(6)
	-2(2)	-2(2)	-2(2)	-2(2)		2(4)		2(4)
	0(6)	0(6)	0(6)	0(6)		4(1)		4(1)
1011	2(2)	2(2)	2(2)	2(2)	-4(1)	-4(2)	-4(1)	-4(2)
1111	2.8284(2)	2.8284(2)	2.8284(2)	2.8284(2)	-2(4)	0(12)	-2(4)	0(12)
	4(2)	4(2)	4(2)	4(2)	0(6)	4(2)	0(6)	4(2)
					2(4)		2(4)	
					4(1)		4(1)	

Table 2. Spectrum of the 4-cell 60/102 NBCA based regular graph

No. of Union (t)	Complement vector	Degree	First Eigenvalue	Second Eigenvalue	Spectral Gap (g)	g/t
1	1,15	4	4	2	2	2
3	1,3,9,15	8	8	4	4	1.33
5	1,3,5,9,11,15	12	12	2	10	2
7	1,3,5,7,9,11,13,15	16	16	0	16	2.2857

Table 3. Comparison of Mukhopadhyay's spectral gaps with our spectral gaps

No. of Union (t)	g/t(Mukhopadhyay's method)	g/t(Our method)
1	0.76	2
3	1.03	1.33
5	1.14	2
7	1.54	2.2857

The following algorithm shows computing the four neighbors of a vertex in G which is the union of G_1 and G_2.

Algorithm. Computing neighbors of a vertex in G
Input: Complement vectors $(F_1, F_2) \in K$ and a state $\mathbf{x} \in G$.
Output: The four neighbors (S_1, S_2, P_1, P_2) of $\mathbf{x}$.
Step 1: Find the next state S_1 (resp. S_2) of $\mathbf{x}$ using the operator $\overline{T}_1$ (resp. $\overline{T}_2$).

$$S_1 = \overline{T}_1\mathbf{x} = T\mathbf{x} \oplus F_1$$
$$S_2 = \overline{T}_2\mathbf{x} = T\mathbf{x} \oplus F_2$$

/* Find the immediate predecessor P_1 (resp. P_2) by using Theorem 4.4 in Step 2 and Step 3 */
Step 2: Compute $W := \mathbf{x} \oplus F_1$ and $V := \mathbf{x} \oplus F_2$.
Step 3: For $W = (w_1, w_2, \cdots, w_n)$ and $V = (v_1, v_2, \cdots, v_n)$, find $P_1 := (p_{11}, p_{12}, \cdots, p_{1n})$ and $P_2 := (p_{21}, p_{22}, \cdots, p_{2n})$

$$p_{11} = w_1, \;\; p_{1n} = w_n, \;\; p_{1k} = w_k \oplus p_{1k+1}$$

$$p_{21} = v_1, \;\; p_{2n} = v_n, \;\; p_{2k} = v_k \oplus p_{2k+1}$$

where $k = 2, \cdots, n-1$.

In general the description of an expander d-regular graph grows exponentially with the number of vertices as the increase of the size of 60/102 NBCA. However as we require to store only two complement vectors F_1 and F_2, this problem is solved by the above algorithm.

5 Conclusion

In this paper, we proposed a method to generate expander graphs with good expansion properties based on group $60/102$ NBCA. The expansion properties by our method is better than the expansion properties proposed by Mukhopadhyay et al.

References

[1] P. Pal Chaudhuri, D. Roy Chowdhury, S. Nandi, and S. Chattopadhyay. Additive cellular automata theory and its application i, ieee computer society press, california. *IEEE Computer Society Press, California*, 2000.

[2] J. Cheeger. A lower bound for the smallest eigenvalue of the laplacian. in problems in analysis(papers dedicated to solomon bochner, 1969, 195-199). *Princeton Univ. Press*, 1970.

[3] S.J. Cho, U.S. Choi, H.D. Kim, and Y.H. Hwang. Analysis of complemented ca derived from linear hybrid group ca, computers and mathematics with applications. *Computers and Mathematics with Applications*, 53:54–63, 2007.

[4] S.J. Cho, U.S. Choi, H.D. Kim, Y.H. Hwang, J.G. Kim, and S.H. Heo. New synthesis of one-dimensional 90/150 linear hybrid group cellular automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26:1720–1724, 2007.

[5] W. Diffie and M. Hellman. New direction in cryptography. *IEEE Transaction on Information Theory*, pages 644–654, 1976.

[6] D.Peleg and E.Upfal. Constructing disjoint paths on expander graphs. *Combinatorica*, pages 289–313, 1989.

[7] O. Goldreich. Candidate one-way functions based on expander graphs. *Cryptology ePrint Archieve, Report 200/063*, pages 1–9, 2000.

[8] S. Hoory, N. Lindal, and A. Wigderson. Expander graphs and their applications. *Bull. AMS*, 2006.

[9] D. Mukhopadhyay and D.R. Chowdhury. Generation of expander graphs using cellular automata and its applications to cryptography. *LNCS*, 4173:636–645, 2006.

[10] M.S. Pinsker. On the complexity of a concentrator. *In 7th International Telegraffic Conference*, pages 1–4, 1973.

[11] M. Sipser and D. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42:1710–1722, 1996.

[12] S. Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55:601–644, 1983.

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 31–44

Probabilistic initial value problem for cellular automaton rule 172

Henryk Fukś [†]

Department of Mathematics, Brock University, St. Catharines, ON L2S 3A1, Canada

We present a method of solving of the probabilistic initial value problem for cellular automata (CA) using CA rule 172 as an example. For a disordered initial condition on an infinite lattice, we derive exact expressions for the density of ones at arbitrary time step. In order to do this, we analyze topological structure of preimage trees of finite strings of length 3. Level sets of these trees can be enumerated directly using classical combinatorial methods, yielding expressions for the number of n-step preimages of all strings of length 3, and, subsequently, probabilities of occurrence of these strings in a configuration obtained from the initial one after n iterations of rule 172. The density of ones can be expressed in terms of Fibonacci numbers, while expressions for probabilities of other strings involve Lucas numbers. Applicability of this method to other CA rules is briefly discussed.

Keywords: cellular automata, initial value problem, preimage trees

1 Introduction

While working on a certain problem in *complexity engineering*, that is, trying to construct a cellular automaton rule performing some useful computational task, the author encountered the following question. Let $f : \{0,1\}^3 \to \{0,1\}$ be defined as

$$f(x_1,x_2,x_3) = \begin{cases} x_2 & \text{if } x_1 = 0, \\ x_3 & \text{if } x_1 = 1. \end{cases} \tag{1}$$

This function may be called *selective copier*, since it returns (copies) one of its inputs x_2 or x_3 depending on the state of the first input variable x_1. Suppose now that s be a bi-infinite sequence of binary symbols, i.e., $s = \ldots s_{-2}s_{-1}s_0s_1s_2\ldots$, $i \in \mathbb{Z}$. We will transform this string using the selective copier, that is, for each i, we keep s_i if it is preceded by 0, or replace it by s_{i+1} otherwise, so that each s_i is simultaneously replaced by $f(s_{i-1}, s_i, s_{i+1})$. Consider now the question: *Assuming that the initial sequence is randomly generated, what is the proportion of 1's in the sequence after n iterations of the aforementioned procedure?*

[†]The author acknowledges partial financial support from Natural Sciences and Engineering Research Counclil of Canada, in the form of a Discovery Grant.

Function defined by eq. (1) is a local function of cellular automaton rule 172, using Wolfram numbering, and the aforementioned question is an example of a broader class of problems, which could be called probabilistic initial value problems for cellular automata: given initial distribution of *infinite configurations*, what is the probability of occurrence of a given *finite string* in a configuration obtained from the initial one by n iterations of the cellular automaton rule? In what follows, we will demonstrate how one can approach probabilistic initial value problem using cellular automaton rule 172 as an example.

2 Basic definitions

Let $\mathcal{G} = \{0, 1, ...N-1\}$ be called *a symbol set*, and let $\mathcal{S}(\mathcal{G})$ be the set of all bisequences over $\mathcal{G}$, where by a bisequence we mean a function on $\mathbb{Z}$ to $\mathcal{G}$. Set $\mathcal{S}(\mathcal{G})$ will be called *the configuration space*. Throughout the remainder of this text we shall assume that $\mathcal{G} = \{0, 1\}$, and the configuration space $\mathcal{S}(\mathcal{G}) = \{0, 1\}^{\mathbb{Z}}$ will be simply denoted by $\mathcal{S}$.

A block of length n is an ordered set $b_0 b_1 \dots b_{n-1}$, where $n \in \mathbb{N}$, $b_i \in \mathcal{G}$. Let $n \in \mathbb{N}$ and let $\mathcal{B}_n$ denote the set of all blocks of length n over $\mathcal{G}$ and $\mathcal{B}$ be the set of all finite blocks over $\mathcal{G}$.

For $r \in \mathbb{N}$, a mapping $f : \{0, 1\}^{2r+1} \mapsto \{0, 1\}$ will be called *a cellular automaton rule of radius* r. Alternatively, the function f can be considered as a mapping of $\mathcal{B}_{2r+1}$ into $\mathcal{B}_0 = \mathcal{G} = \{0, 1\}$.

Corresponding to f (also called *a local mapping*) we define a *global mapping* $F : \mathcal{S} \to \mathcal{S}$ such that $(F(s))_i = f(s_{i-r}, \dots, s_i, \dots, s_{i+r})$ for any $s \in \mathcal{S}$.

A *block evolution operator* corresponding to f is a mapping $\mathbf{f} : \mathcal{B} \mapsto \mathcal{B}$ defined as follows. Let $r \in \mathbb{N}$ be the radius of f, and let $a = a_0 a_1 \dots a_{n-1} \in \mathcal{B}_n$ where $n \geq 2r + 1 > 0$. Then

$$\mathbf{f}(a) = \{f(a_i, a_{i+1}, \dots, a_{i+2r})\}_{i=0}^{n-2r-1}. \tag{2}$$

Note that if $b \in B_{2r+1}$ then $f(b) = \mathbf{f}(b)$.

We will consider the case of $\mathcal{G} = \{0, 1\}$ and $r = 1$ rules, i.e., *elementary cellular automata*. In this case, when $b \in \mathcal{B}_3$, then $f(b) = \mathbf{f}(b)$. The set $\mathcal{B}_3 = \{000, 001, 010, 011, 100, 101, 101, 110, 111\}$ will be called the set of *basic blocks*.

The number of n-step preimages of the block b under the rule f is defined as the number of elements of the set $\mathbf{f}^{-n}(b)$. Given an elementary rule f, we will be especially interested in the number of n-step preimages of basic blocks under the rule f.

3 Probabilistic initial value problem

The appropriate mathematical description of an initial distribution of configurations is a probability measure μ on $\mathcal{S}$. Such a measure can be formally constructed as follows. If b is a block of length k, i.e., $b = b_0 b_1 \dots b_{k-1}$, then for $i \in \mathbb{Z}$ we define a cylinder set. The cylinder set is a set of all possible configurations with fixed values at a finite number of sites. Intuitively, measure of the cylinder set given by the block $b = b_0 \dots b_{k-1}$, denoted by $\mu[C_i(b)]$, is simply a probability of occurrence of the block b in a place starting at i. If the measure μ is shift-invariant, than $\mu(C_i(b))$ is independent of i, and we will therefore drop the index i and simply write $\mu(C(b))$.

The Kolmogorov consistency theorem states that every probability measure μ satisfying the consistency condition

$$\mu[C_i(b_1 \dots b_k)] = \mu[C_i(b_1 \dots b_k, 0)] + \mu[C_i(b_1 \dots b_k, 1)] \tag{3}$$

extends to a shift invariant measure on $\mathcal{S}$ (Dynkin, 1969) .For $p \in [0,1]$, the Bernoulli measure defined as $\mu_p[C(b)] = p^j(1-p)^{k-j}$, where j is a number of ones in b and $k-j$ is a number of zeros in b, is an example of such a shift-invariant (or spatially homogeneous) measure. It describes a set of random configurations with the probability p that a given site is in state 1.

Since a cellular automaton rule with global function F maps a configuration in $\mathcal{S}$ to another configuration in $\mathcal{S}$, we can define the action of F on measures on $\mathcal{S}$. For all measurable subsets E of $\mathcal{S}$ we define $(F\mu)(E) = \mu(F^{-1}(E))$, where $F^{-1}(E)$ is an inverse image of E under F.

If the initial configuration was specified by μ_p, what can be said about $F^n\mu_p$ (i.e., what is the probability measure after n iterations of F)? In particular, given a block b, what is the probability of the occurrence of this block in a configuration obtained from a random configuration after n iterations of a given rule?

The general question of finding the iterrates of the Bernoulli measure under a given CA has been extensively studied in recent years by many authors, including, among others, Lind (1984); Ferrari et al. (2000); Maass and Martínez (2003); Host et al. (2003); Pivato and Yassawi (2002, 2004); Maass et al. (2006) and Maass et al. (2006). In this paper, we will approach the problem from somewhat different angle, using very elementary methods and without resorting to advanced apparatus of ergodic theory and symbolic dynamics. We will consider iterates of the Bernoulli measure by analyzing patterns in preimage sets.

For a given block b, the set of n-step preimages is $\mathbf{f}^{-n}(b)$. Then, by the definition of the action of F on the initial measure, we have

$$(F^n\mu_p)(C(b)) = \mu_p\left(F^{-n}(C(b))\right), \tag{4}$$

and consequently

$$(F^n\mu_p)(C(b)) = \sum_{a\in\mathbf{f}^{-n}(b)} \mu_p(a). \tag{5}$$

Let us define the probability of occurrence of block b in a configuration obtained from the initial one by n iterations of the CA rule as

$$P_n(b) = (F^n\mu_p)(C(b)). \tag{6}$$

Using this notation, eq. (5) becomes

$$P_n(b) = \sum_{a\in\mathbf{f}^{-n}(b)} P_0(a). \tag{7}$$

If the initial measure is $\mu_{1/2}$, then all blocks of a given length are equally probable, and $P_0(a) = \frac{1}{2^{|a|}}$, where $|a|$ is the length of the block a. For elementary CA rule, the length of n-step preimage of b is $2n+|b|$, therefore

$$P_n(b) = 2^{-|b|-2n}\operatorname{card}\mathbf{f}^{-n}(b). \tag{8}$$

This equation tells us that if the initial measure is symmetric ($\mu_{1/2}$), then all we need to know in order to compute $P_n(b)$ is the cardinality of $\mathbf{f}^{-n}(b)$. One way to think about this is to draw a *preimage tree* for b. We start form b as a root of the tree, and determine all its preimages. Then each of these preimages is connected with b by an edge. They constitute level 1 of the preimage tree. Then, for each block of level 1, we again compute its preimages and we link them with that block, thus obtaining level 2. Repeating this operation *ad infinitum*, we obtain a tree such as the one shown in Figure 1. In that figure, five levels of the preimage tree for rule 172 rooted at 101 are shown, with only first level labelled.

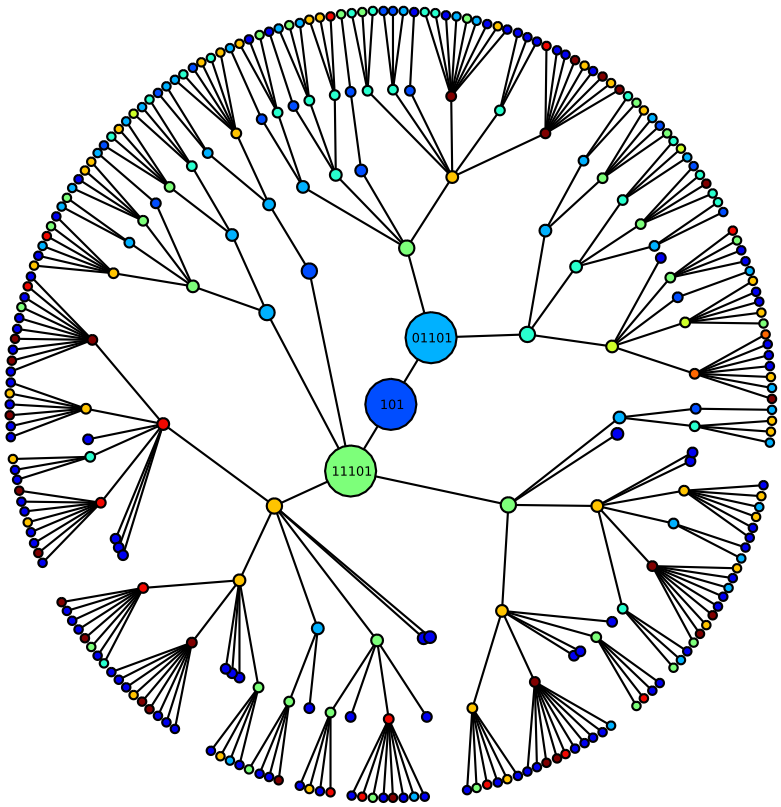


Fig. 1: Preimage tree for rule 172 rooted at 101.

Note that $\operatorname{card}\mathbf{f}^{-n}(b)$ corresponds to the number of vertices in the n-th level of the preimage tree. One thus only needs to know cardinalities of level sets in order to use eq. (8), while the exact topology of connections between vertices of the preimage tree is unimportant. The key problem, therefore, is to enumerate level sets. In order to answer the question posed in the introduction, we need to compute $P_n(1)$ for rule 172, which, in turn, requires that we enumerate level sets of a preimage tree rooted at 1. It turns out that for rule 172 the preimage tree rooted at 1 is rather complicated, and that it is more convenient to consider preimage trees rooted at other blocks. In the next section, we will show how to express $P_n(1)$ by some other block probabilities. From now on, $\mathbf{f}$ will exclusively denote the block evolution operator for rule 172.

4 Block probabilities

Since $\mathbf{f}^{-1}(1) = \{010, 011, 101, 111\}$, we have $P_{n+1}(1) = P_n(010) + P_n(011) + P_n(101) + P_n(111)$. Due to consistency conditions (eq. 3), $P_n(010) + P_n(011) = P_n(01)$, and we obtain

$$P_{n+1}(1) = P_n(01) + P_n(101) + P_n(111). \tag{9}$$

This can be transformed even further by noticing that $P_n(01) = P_n(001) + P_n(101)$, therefore

$$P_n(1) = P_{n-1}(001) + 2P_{n-1}(101) + P_{n-1}(111). \tag{10}$$

By using eq. (8) and defining $c_n = P_n(1)$ we obtain

$$c_n = \frac{\operatorname{card} \mathbf{f}^{-n+1}(001) + 2\operatorname{card} \mathbf{f}^{-n+1}(101) + \operatorname{card} \mathbf{f}^{-n+1}(111)}{2^{2n+1}}. \tag{11}$$

This means that in order to compute c_n, we need to know cardinalities of n-step preimages of 001, 101, and 111.

5 Structure of preimage sets

The structure of level sets of preimage trees rooted at 001, 101, and 111 will be described in the following three propositions.

Proposition 5.1 *Block b belongs to $\mathbf{f}^{-n}(001)$ if and only if it has the structure*

$$b = \underbrace{\star\star\ldots\star}_{n}001\underbrace{\star\star\ldots\star}_{n}, \tag{12}$$

where $\star$ represents arbitrary symbol from the set $\{0, 1\}$.

Let us first observe that $\mathbf{f}^{-1}(001) = \{00010, 00011, 10010, 10011\}$, which means that $\mathbf{f}^{-1}(001)$ can be represented as $\star\star001\star\star$. Similarly, therefore, $\mathbf{f}^{-2}(001)$ has the structure $\star\star\star001\star\star\star$, and by induction, for any n, the structure of $\mathbf{f}^{-n}(001)$ must be $\underbrace{\star\star\ldots\star}_{n}001\underbrace{\star\star\ldots\star}_{n}$. □

Proposition 5.2 *Block b belongs to $\mathbf{f}^{-n}(101)$ if and only if it has the structure*

$$b = \underbrace{\star\star\ldots\star}_{n-1}a_1a_2\ldots a_n1101, \tag{13}$$

where $a_i \in \{0, 1\}$ for $i = 1, \ldots, n$ and the string $a_1a_2\ldots a_n$ does not contain any pair of adjacent zeros, that is. $a_ia_{i+1} \neq 00$ for all $i = 1, \ldots, n-1$.

Two observations will be crucial for the proof. First of all, $\mathbf{f}^{-1}(101) = \{01101, 11101\}$, thus $\mathbf{f}^{-1}(101)$ has the structure $\star1101$. Furthermore, we have $\mathbf{f}^{-1}(1101) = \{011101, 101101, 111101\}$, meaning that if 1101 appears in a configuration, and is not preceded by 00, then after application of the rule 172, 1101 will still appear, but shifted one position to the left. All this means that if b is to be an n-step preimage of 101, it must end with 1101. After each application of rule 172 to b, the block 1101 will remain at the end as long as it is not preceded by two zeros.

Now, let us note that $\mathbf{f}^{-1}(00) = \{0000, 0001, 1000, 1001, 1100\}$, which means that preimage of 00 is either 1100 or $\star00\star$. Therefore, we can say that if 00 is not present in the string $a_1a_2\ldots a_n$, it will not appear in its consecutive images under $\mathbf{f}$. Thus, block 1101 will, after each iteration of f, remain at the end, and will never be preceded by two zeros. Eventually, after n iterations, it will produce 101, as shown

in the example below.

$$\begin{array}{cccccccccccccc}
1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \\
 & & 0 & 0 & 0 & 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & & \\
 & & & 0 & 0 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & & & \\
 & & & & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & & & & \\
 & & & & & 1 & 0 & 1 & & & & &
\end{array}$$

What is left to show is that not having 00 in $a_1a_2\ldots a_n$ is necessary. This is a consequence of the fact that $\mathbf{f}(\star 00\star) = 00$, which means that if 00 appears in a string, then it stays in the same position after the rule 172 is applied. Indeed, if we had a pair of adjacent zeros in $a_1a_2\ldots a_n$, it would stay in the same position when $\mathbf{f}$ is applied, and sooner or later block 1101, which is moving to the left, would come to the position immediately following this pair, and would be destroyed in the next iteration, thus never producing 101. Such a process is illustrated below, where after three iterations the block 1101 is destroyed due to "collision" with 00. □

$$\begin{array}{cccccccccccccc}
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\
 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \\
 & & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & & \\
 & & & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & & & \\
 & & & & 0 & 0 & 1 & 0 & 1 & & & & \\
 & & & & & 0 & 1 & 1 & & & & &
\end{array}$$

Proposition 5.3 *Block b belongs to $\mathbf{f}^{-n}(111)$ if and only if it has the structure*

$$b = \underbrace{\star\star\ldots\star}_{n-2} a_1a_2\ldots a_{n+5}, \tag{14}$$

where $a_i \in \{0,1\}$ for $i = 1,\ldots,n$ and the string $a_1a_2\ldots a_n$ satisfies the following three conditions:

(i) $a_ia_{i+1} \neq 00$ *for all* $i = 2\ldots n+4$*;*

(ii) $a_{n+3}a_{n+4}a_{n+5} \neq 110$ *and* $a_{n+2}a_{n+3}a_{n+4} \neq 110$*;*

(iii) if $a_1a_2 \neq 00$*, then* $a_{n+1}a_{n+2}a_{n+3} \neq 110$*.*

We will present only the main idea of the proof here, omitting some tedious details. It will be helpful to inspect spatiotemporal pattern generated by rule 172 first, as shown in Figure 2. Careful inspection of this pattern reveals three facts, each of them easily provable in a rigorous way:

(F1) A cluster of two or more zeros keeps its right boundary in the same place for ever.

(F2) A cluster of two or more zeros extends its left boundary to the left one unit per time step as long as the left boundary is preceded by two or more ones. If the left boundary it is preceded by 01, it stays in the same place.

(F3) Isolated zero moves to the left one step at a time as long as it has at least two ones on the left. If an isolated zero is preceded by 10, it disappears in the next time step.

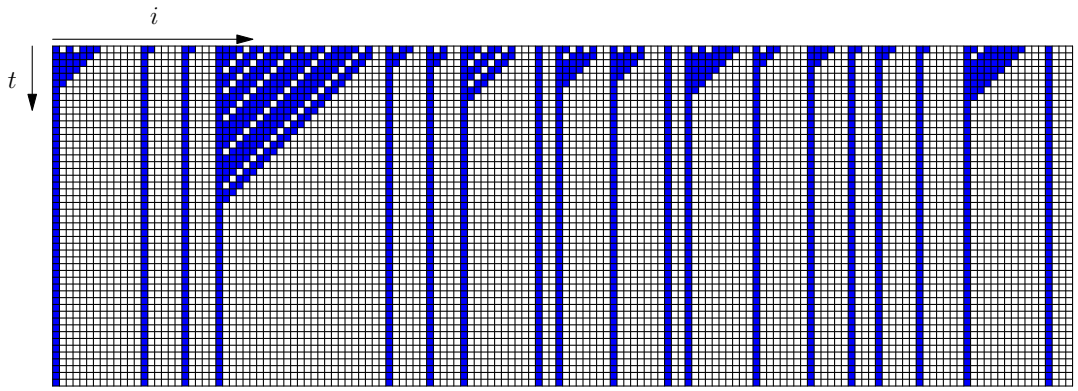


Fig. 2: Example of a spatiotemporal pattern produced by rule 172.

Let us first prove that (i)-(iii) are necessary. Condition (i) is needed because if we had 00 in the string $a_2 \dots a_{n+5}$, its left boundary would grow to the left and after n iterations it would reach sites in which we expect to find the resulting string 111.

Moreover, string $a_1 a_2 \dots a_{n+5}$ cannot have 011 at the end position, one site before the end, or two sites before the end. If it had, 0 preceded by two 1's would move to the left and, after n iterations, it would reach sites where we want to find 111. The only exception to this is the case when $a_0 a_1 = 00$. In this case, even if 011 is in the second position from the end, it will disappear in step $n-1$. This demonstrates that (ii) and (iii) are necessary.

In order to prove sufficiency of (i)-(iii), let us suppose that the string b satisfies all these conditions yet $\mathbf{f}^n(b) \neq 111$. This would imply that at least one of the symbols of $\mathbf{f}^n(b)$ is equal to zero. However, according to what we stated in F1–F3, zero can appear in a later configuration only as a result of growth of an initial cluster of two of more zeros, or by moving to the left if it is preceded by two ones. This, however, is impossible due to conditions (i)-(iii). □.

6 Enumeration of preimage strings

Once we know the structure of preimage sets, we can enumerate them. For this, the following lemma will be useful.

Lemma 6.1 *The number of binary strings $a_1 a_2 \dots a_n$ such that 00 does not appear as two consecutive terms $a_i a_{i+1}$ is equal to F_{n+2}, where F_n is the n-th Fibonacci number.*

This result will be derived using classical transfer-matrix method. Let $g(n)$ be the number of binary strings $a_1 a_2 \dots a_n$ such that 00 does not appear as two consecutive terms $a_i a_{i+1}$. We can think of such string as a walk of length n on a graph with vertices $v_1 = 0$ and $v_2 = 1$ which has adjacency matrix A given by $A_{11} = 0$, $A_{12} = A_{21} = A_{22} = 1$. One can prove that the generating function for g,

$$G(\lambda) = \sum_{n=0}^{\infty} g(n+1)\lambda^n, \tag{15}$$

can be expressed by $G(\lambda) = G_{11}(\lambda) + G_{12}(\lambda) + G_{21}(\lambda) + G_{22}(\lambda)$, where

$$G_{ij} = \frac{(-1)^{i+j} \det(I - \lambda A : j, i)}{\det(I - \lambda A)}, \tag{16}$$

and where $(M : j, i)$ denotes the matrix obtained by removing the $j - th$ row and $i - th$ column of M. Proof of this statement can be found, for example, in Stanley (1986). Applying this to the problem at hand we obtain

$$G(\lambda) = \frac{-(2 + \lambda)}{-1 + \lambda + \lambda^2}. \tag{17}$$

By decomposing the above generating function into simple fractions we get

$$G(\lambda) = \frac{\frac{3}{10}\sqrt{5} - \frac{1}{2}}{\lambda + \psi} + \frac{-\frac{1}{2} - \frac{3}{10}\sqrt{5}}{\lambda + 1 - \psi}, \tag{18}$$

where $\psi = \frac{1}{2} + \frac{1}{2}\sqrt{5}$ is the golden ratio. Now, by using the fact that

$$\frac{1}{\lambda + \psi} = -\sum_{n=0}^{\infty} \left(\frac{-1}{\psi}\right)^{n+1} \lambda^n, \tag{19}$$

and by using a similar expression for $\frac{1}{\lambda+1-\psi}$, we obtain

$$G(\lambda) = \sum_{n=0}^{\infty} F_{n+3} \lambda^n, \tag{20}$$

where F_n is the n-th Fibonacci number, $F_n = \dfrac{\psi^n - (1 - \psi)^n}{\sqrt{5}}$. This implies that $g(n) = F_{n+2}$. □

Proposition 6.1 *The cardinalities of preimage sets of* 001*,* 100*,* 101 *and* 111 *are given by*

$$\operatorname{card} \mathbf{f}^{-n}(001) = 4^n, \tag{21}$$

$$\operatorname{card} \mathbf{f}^{-n}(101) = 2^{n-1} F_{n+2}, \tag{22}$$

$$\operatorname{card} \mathbf{f}^{-n}(111) = 2^n F_{n+3}. \tag{23}$$

Proof of the first of these formulae is a straightforward consequence of Proposition 5.1. We have $2n$ arbitrary binary symbols in the string b, thus the number of such strings must be $2^{2n} = 4^n$.

The second formula can be immediately obtained using Lemma 6.1 and Proposition 5.1. Since the first $n - 1$ symbols of $\mathbf{f}^{-n}(101)$ are arbitrary, and the remaining symbols form a sequence of n symbols without 00, we obtain

$$\operatorname{card} \mathbf{f}^{-n}(101) = 2^{n-1} F_{n+2}. \tag{24}$$

In order to prove the third formula, we will use Proposition 5.3. We need to compute the number of binary strings $a_1 a_2 \ldots a_{n+5}$ satisfying conditions (i)-(iii) of Proposition 5.3. Le us first introduce a symbol $\alpha_1 \alpha_2 \ldots \alpha_k$ to denote the string of length k in which no pair 00 appears. Then we define:

- A is the set of all strings having the form $\alpha_1 \alpha_2 \ldots \alpha_{n+5}$,

- A_1 is the set of all strings having the form $\alpha_1\alpha_2\ldots\alpha_{n+2}110$,
- A_2 is the set of all strings having the form $\alpha_1\alpha_2\ldots\alpha_{n+1}1101$,
- A_3 is the set of all strings having the form $\alpha_1\alpha_2\ldots\alpha_n11010$,
- A_4 is the set of all strings having the form $\alpha_1\alpha_2\ldots\alpha_n11011$,
- B is the set of all strings having the form $001\alpha_1\alpha_2\ldots\alpha_{n+2}$,
- B_1 is the set of all strings having the form $001\alpha_1\alpha_2\ldots\alpha_{n-1}110$,
- B_2 is the set of all strings having the form $001\alpha_1\alpha_2\ldots\alpha_{n-2}1101$.

The set Ω of binary strings $a_1a_2\ldots a_{n+5}$ satisfying conditions (i)-(iii) of Proposition 5.3 can be now written as

$$\Omega = A \setminus (A_1 \cup A_2 \cup A_3 \cup A_4) \cup B \setminus (B_1 \cup B_2). \tag{25}$$

Since $A_1 \ldots A_4$ are mutually disjoint, and B_1 and B_2 are disjoint too, the number elements in the set Ω is

$$\begin{aligned}\operatorname{card}\Omega = \operatorname{card} A - \operatorname{card} A_1 - \operatorname{card} A_2 - \operatorname{card} A_3 - \operatorname{card} A_4 \\ + \operatorname{card} B - \operatorname{card} B_1 - \operatorname{card} B_2,\end{aligned} \tag{26}$$

which, using Lemma 6.1, yields

$$\operatorname{card}\Omega = F_{n+7} - (F_{n+4} + F_{n+3} + F_{n+2} + F_{n+2}) + F_{n+4} - (F_{n+1} + F_n). \tag{27}$$

Using basic properies of Fibonacci numbers, the above simplifies to $\operatorname{card}\Omega = 4F_{n+3}$. Now, since in the Proposition 5.3 the string $a_1\ldots a_{n+5}$ is preceded by $n-2$ arbitrary symbols, we obtain

$$\operatorname{card}\mathbf{f}^{-n}(111) = 2^{n-2}\cdot 4F_{n+3} = 2^nF_{n+3}, \tag{28}$$

what was to be shown.

7 Density of ones

Using results of the previous section, eq. (11) can now be rewritten as

$$c_n = \frac{4^{n-1} + 2^{n-1}F_{n+1} + 2^{n-1}F_{n+2}}{2^{2n+1}}, \tag{29}$$

which simplifies to

$$c_n = \frac{1}{8} + \frac{F_{n+3}}{2^{n+2}}, \tag{30}$$

or, more explicitly, to

$$c_n = \frac{1}{8} + \frac{(1+\sqrt{5})^{n+3} - (1-\sqrt{5})^{n+3}}{2^{2n+5}\sqrt{5}}. \tag{31}$$

Obviously, $\lim_{n\to\infty} c_n = \frac{1}{8}$, in agreement with the numerical value reported in Wolfram (1994). We can see that c_n converges toward c_∞ exponentially fast, with some damped oscillations superimposed over

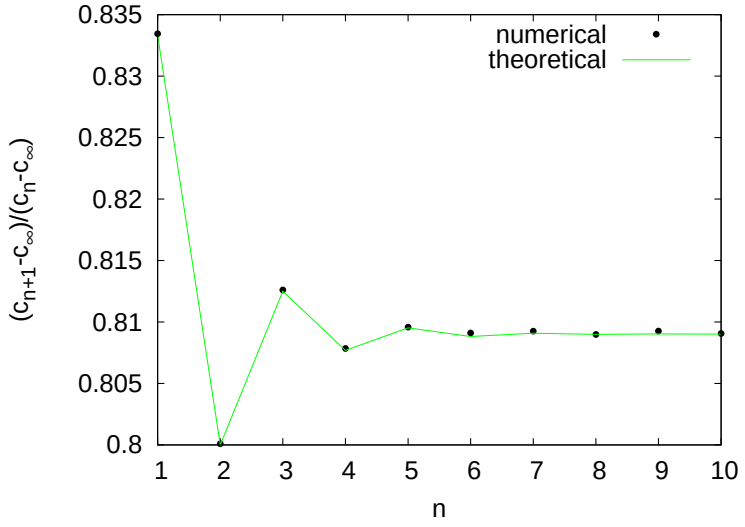


Fig. 3: Plot of the ratio $\frac{c_{n+1}-c_\infty}{c_n-c_\infty}$ as a function of time step n. Numerical results were obtained by iterating rule 172 on a a configuration of length 10^8 with periodic boundary conditions.

the exponential decay. This is illustrated in Figure 3, where, in order to emphasize the aforementioned oscillations, instead of c_n we plotted the ratio

$$d_n = \frac{c_{n+1}-c_\infty}{c_n-c_\infty} \tag{32}$$

as a function of n. One can show that d_n converges to the half of *ratio divina* (golden ratio), $\psi/2 \approx 0.809016\ldots$, as illustrated in Figure 3. We can see from this figure that the convergence is very fast and that the agreement between numerical simulations and the theoretical formula is nearly perfect.

8 Further results

Results obtained in the previous two sections suffice to compute block probabilities for all blocks of length up to 3. Proposition 6.1 together with eq. (8) yields formulas for $P_n(001)$, $P_n(101)$, and $P_n(111)$. Consistency conditions give $P_n(01) = P_n(001) + P_n(101)$. Furthermore $P_n(10) = P_n(01)$ due to the fact that $P_n(10)+P_n(00) = P_n(01)+P_n(00) = P_n(0)$. Applying consistency conditions again we have $P_n(1) = P_n(10)+P_n(11)$, hence $P_n(11) = P_n(1)-P_n(10)$, and, similarly, $P_n(00) = P_n(0)-P_n(10)$. This gives us probabilities of all blocks of length 2. Probabilities of blocks of length 3 can be obtained in

a similar fashion:

$$\begin{aligned}
P_n(000) &= P_n(00) - P_n(100),\\
P_n(110) &= P_n(11) - P_n(111),\\
P_n(011) &= P_n(11) - P_n(111),\\
P_n(010) &= P_n(01) - P_n(011).
\end{aligned}$$

The only missing probability, $P_n(100)$ is the same as $P_n(001)$ because $P_n(100)+P_n(000) = P_n(001)+P_n(000) = P_n(00)$. The following formulas summarize these results.

$$\begin{aligned}
P_n(000) &= 5/8 - 2^{-n-2}F_{n+3} - 2^{-n-4}F_{n+2},\\
P_n(001) &= 1/8,\\
P_n(010) &= 1/8 - 2^{-n-3}F_{n+1},\\
P_n(011) &= 2^{-n-4}L_{n+2},\\
P_n(100) &= 1/8,\\
P_n(101) &= 2^{-n-4}F_{n+2},\\
P_n(110) &= 2^{-n-4}L_{n+2},\\
P_n(111) &= 2^{-n-3}F_{n+3},
\end{aligned}$$

where $L_n = 2F_{n+1} - F_n$ is the n-th Lucas number. We can also rewrite these formulas in terms of cardinalities of preimage sets using eq. (8), as stated below.

Theorem 8.1 *Let* $\mathbf{f}$ *be the block evolution operator for CA rule 172. Then for any positive integer* n *we have*

$$\begin{aligned}
\operatorname{card}\mathbf{f}^{-n}(000) &= 5\cdot 4^n - 2^{n+1}F_{n+3} - 2^{n-1}F_{n+2},\\
\operatorname{card}\mathbf{f}^{-n}(001) &= 4^n,\\
\operatorname{card}\mathbf{f}^{-n}(010) &= 4^n - 2^nF_{n+1},\\
\operatorname{card}\mathbf{f}^{-n}(011) &= 2^{n-1}L_{n+2},\\
\operatorname{card}\mathbf{f}^{-n}(100) &= 4^n,\\
\operatorname{card}\mathbf{f}^{-n}(101) &= 2^{n-1}F_{n+2},\\
\operatorname{card}\mathbf{f}^{-n}(110) &= 2^{n-1}L_{n+2},\\
\operatorname{card}\mathbf{f}^{-n}(111) &= 2^nF_{n+3},
\end{aligned}$$

where F_n *is the* n*-th Fibonacci number,* $F_n = \dfrac{\psi^n - (1-\psi)^n}{\sqrt{5}}$, $\psi = \frac{1}{2} + \frac{1}{2}\sqrt{5}$, *and* L_n *is the* n*-th Lucas number,* $L_n = \psi^n + (1-\psi)^n$.

9 Concluding remarks

The method for computing block probabilities in cellular automata described in this paper is certainly not applicable to arbitrary CA rule. It will work only if the structure of level sets of preimage trees is

sufficiently regular so that the level sets can be enumerated by some known combinatorial technique. Altough "chaotic" rules like rule 18, or complex rules such as rule 110 certainly do not belong to this category, in surprisingly many cases significant regularities can be detected in preimage trees. Usually, this applies to "simple" rules, those which in Wolfram classification belong to class I, class II, and sometimes class III. Rule 172 reported here is one of the most interesting among such rules, primarily because the density of ones does not converge exponentially to some fixed value as in many other cases, but exhibits subtle damped oscillations on top of the exponential decay. Furthermore, the appearance of Fibonacci and Lucas numbers in formulas for block probabilities is rather surprising.

One should add at this point that the convergence toward the steady state can be slower than exponential even in fairly "simple" cellular automata. Using similar method as in this paper, it has been found in Fukś and Haroutunian (2009) that in rule 14 the density of ones converges toward its limit value approximately as a power law. The exact formula for the density of ones in rule 14 involves Catalan numbers, and the structure of level sets is quite different than the one reported here. Rule 142 exhibits somewhat similar behavior too, as reported in Fukś (2006).

As a final remark, let us add that the results presented here assume initial measure $\mu_{1/2}$. This can be generalized to arbitrary μ_p. In order to do this, one needs, instead of straightforward counting of preimages, to perform direct computation of their probabilities using methods based on Markov chain theory. Work on this problem is ongoing and will be reported elsewhere.

References

E. B. Dynkin. *Markov Processes-Theorems and Problems*. Plenum Press, New York, 1969.

P. A. Ferrari, A. Maass, S. Martínez, and P. Ney. Cesàro mean distribution of group automata starting from measures with summable decay. *Ergodic Theory Dynam. Systems*, 20(6):1657–1670, 2000.

H. Fukś. Dynamics of the cellular automaton rule 142. *Complex Systems*, 16:123–138, 2006.

H. Fukś and J. Haroutunian. Catalan numbers and power laws in cellular automaton rule 14. *Journal of cellular automata*, 4:99–110, 2009.

B. Host, A. Maass, and S. Martínez. Uniform Bernoulli measure in dynamics of permutative cellular automata with algebraic local rules. *Discrete Contin. Dyn. Syst.*, 9(6):1423–1446, 2003.

D. A. Lind. Applications of ergodic theory and sofic systems to cellular automata. *Phys. D*, 10(1-2): 36–44, 1984. Cellular automata (Los Alamos, N.M., 1983).

A. Maass and S. Martínez. Evolution of probability measures by cellular automata on algebraic topological markov chains. *Biol. Res.*, 36(1):113–118, 2003.

A. Maass, S. Martínez, M. Pivato, and R. Yassawi. Asymptotic randomization of subgroup shifts by linear cellular automata. *Ergodic Theory and Dynamical Systems*, 26(04):1203–1224, 2006.

A. Maass, S. Martínez, and M. Sobottka. Limit measures for affine cellular automata on topological markov subgroups. *Nonlinearity*, 19:2137–2147, Sept. 2006.

M. Pivato and R. Yassawi. Limit measures for affine cellular automata. *Ergodic Theory Dynam. Systems*, 22(4):1269–1287, 2002.

M. Pivato and R. Yassawi. Limit measures for affine cellular automata. II. *Ergodic Theory Dynam. Systems*, 24(6):1961–1980, 2004.

R. P. Stanley. *Enumerative combinatorics*. Wadsworth and Brooks/Cole, Monterey, 1986.

S. Wolfram. *Cellular Automata and Complexity: Collected Papers*. Addison-Wesley, Reading, Mass., 1994.

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 45–54

Block-sequential update schedules and Boolean automata circuits

Eric Goles[1,2] and Mathilde Noual[3,4†]

[1] *University Adolfo Ibañez, Peñalolen, Santiago, Chile*
[2] *Complex Systems Institute of Valparaiso, ISCV, Valparaiso, Chile*
[3] *University of Lyon, ÉNS-Lyon, LIP, CNRS UMR 5668, F-69007, Lyon, France*
[4] *Rhône-Alpes Complex Systems Institute, IXXI, F-69007, Lyon, France*

Our work is set in the framework of complex dynamical systems and, more precisely, that of Boolean automata networks modeling regulation networks. We study how the choice of an update schedule impacts on the dynamics of such a network. To do this, we explain how studying the dynamics of any network updated with an arbitrary block-sequential update schedule can be reduced to the study of the dynamics of a different network updated in parallel. We give special attention to networks whose underlying structure is a circuit, that is, Boolean automata circuits. These particular and simple networks are known to serve as the "engines" of the dynamics of arbitrary regulation networks containing them as sub-networks in that they are responsible for their variety of dynamical behaviours. We give both the number of attractors of period p, $\forall p \in \mathbb{N}$ and the total number of attractors in the dynamics of Boolean automata circuits updated with any block-sequential update schedule. We also detail the variety of dynamical behaviours that such networks may exhibit according to the update schedule.

Keywords: Boolean automata network, cycles/circuits, attractors, discrete dynamical system, update/iteration schedule

1 Introduction

From the point of view of theoretical biology as well as that of theoretical computer science, it seems to be of great interest to address the question of the number of different asymptotic dynamical behaviours of a regulation network. Close to the 16th Hilbert problem concerning the number of limit cycles of dynamical systems [10], this question has already been considered in a certain number of works [3, 2, 13]. In the same lines and with a similar will to understand the dynamical properties of (regulation) networks, we decided to focus on the dynamics of Boolean automata networks.

Two aspects of these networks caught our attention. The first one is that, as Thomas [15] already noticed, the "driving force" of their dynamics lies in their underlying circuits. Indeed, a network whose underlying interaction graph is an acyclic digraph can only eventually end up in a configuration that will never change over time (*aka.* fixed point). A network with retroactive loops, on the contrary, exhibits

†corresponding author

more diverse dynamical behaviour patterns. This is why, before attempting to explain theoretically the dynamics of Boolean automata networks whose interaction graphs are arbitrary, we decided to pay special attention to the simple instance of Boolean automata networks that are Boolean automata circuits[(i)].

The other essential aspect of Boolean networks, or more generaly, of regulation networks, that we concentrated on is their update schedule, that is, the order according to which the different interactions that define the system occur. Robert [14] highlighted the importance of update schedules on the dynamics of a system. In [7], the focus was put on the parallel update schedule that updates all automata of a network synchronously at each time step of a discretised time scale. Now, although biological knowledge about the precise schedules of gene regulations lack, one may argue reasonably that genes involved in a same cellular physiological function are highly unlikely to perform there regulations in perfect synchrony although biologists seem to agree that a certain amount of synchrony is not, on the whole, implausible. In this paper, we consider a looser version of the parallel update schedule, namely, the general *block-sequential* schedule that updates every automaton of a network exactly once at every step according to a predefined order but which does not impose that all automata be updated at once. In other words, block-sequential schedules define *blocks* of automata to be updated sequentially while within the blocks, the automata are updated synchronously.

Section 2 introduces some definitions relative to general Boolean automata networks as well as some preliminary results. Section 3 focuses on Boolean automata circuits and on their dynamics under arbitrary block-sequential update schedules[(ii)].

2 Networks and their dynamics

We define a *Boolean automata network* of size n as a couple $N = (G, \mathcal{F})$ where $G = (V, A)$ is a digraph of order $|V| = n$ called the *interaction graph* of the network. The nodes of G are assimilated to the automata of N. Vectors of $\{0,1\}^n$ are seen as *configurations* of N. Their i^{th} components are the states of nodes $i \in V$. $\mathcal{F} = \{f_i : \{0,1\}^n \to \{0,1\} \mid i \in V\}$ is the set of *local transition functions* of the network. For each node $i \in V$, and each configuration $x \in \{0,1\}^n$, $f_i(x)$ depends only on the components x_j such that $(j,i) \in A$. For the sake of simplicity, we consider abusingly, in some cases, that f_i is a function of arity $deg^-(i) = |\{j \in V \mid (j,i) \in A\}|$ instead of n.

To define the dynamics of N, an update schedule s of the states of nodes needs to be specified. In this paper, we consider only *block-sequential update schedules*, that is, functions $s : V \to \{0, \dots, n-1\}$ such that for any node $i \in V$, $s(i)$ gives the date of update of node i ($t + \frac{s(i)}{s_{max}}$, $s_{max} = max\{s(i) \mid i \in V\}$) between any two time steps t and $t+1$. Thus, *within* a time step t, the states of all nodes are updated exactly once. Without loss of generality, we suppose that update schedules s impose no "waiting period" within a time step: $min\{s(i) \mid i \in V\} = 0$ and $\forall 0 \leq d < n-1,\ \exists i \in V,\ s(i) = d+1 \Rightarrow \exists j \in V,\ s(j) = d$. The *parallel update schedule* denoted here by π is the update schedule such that $\forall i \in V,\ \pi(i) = 0$. It updates all nodes at once. A *sequential update schedule* is a block-sequential update schedule s that updates only one node at a time: $\forall i \neq j,\ s(i) \neq s(j)$. The number of different update schedules of a set of n elements is known to be exponential in n [6].

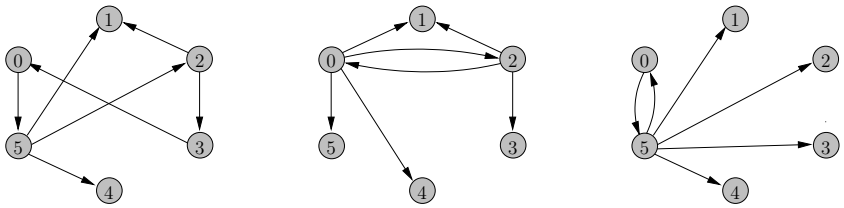
Example 2.1 *Let $V = \{0, \dots, 5\}$. The function $r : V \to \{0, \dots, 5\}$ such that $r(2) = 0$, $r(3) = r(4) = 1$ and $r(0) = r(1) = r(5) = 2$ is a block sequential update schedule. The function $s : V \to \{0, \dots, 5\}$*

(i) and which also happen to be a simple instance of *threshold* Boolean automata networks [11].

(ii) Results presented in this paper and their proofs are detailed in [12].

such that $s(5) = 0,\ s(3) = 1,\ s(1) = 2,\ s(0) = 3,\ s(2) = 4$ *and* $s(4) = 5$ *is a sequential update schedule. A more practical way of denoting* r*,* s *and the parallel update schedule is the following:*

$$r \equiv (2)(3,4)(0,1,5) \qquad s \equiv (5)(3)(1)(0)(2)(4) \qquad \pi \equiv (0,1,2,3,4,5).$$



$G = G^{\pi}$ $\qquad$ G^{s} $\qquad$ G^{r}

i	$x_i(t+1)$		
$\in V$	π	$s = (5)(3)(1)(0)(2)(4)$	$r = (2)(3,4)(0,1,5)$
0	$f_0(x_3(t))$	$f_0(x_3(t+1))$ $= f_0 \circ f_3(x_2(t))$	$f_0(x_3(t+1))$ $= f_0 \circ f_3 \circ f_2(x_5(t))$
1	$f_1(x_2(t), x_5(t))$	$f_1(x_2(t), x_5(t+1))$ $= f_1(x_2(t), f_5(x_0(t)))$	$f_1(x_2(t+1), x_5(t))$ $= f_1(f_2(x_5(t)), x_5(t))$
2	$f_2(x_5(t))$	$f_2(x_5(t+1))$ $= f_2 \circ f_5(x_0(t))$	$f_2(x_5(t))$
3	$f_3(x_2(t))$	$f_3(x_2(t))$	$f_3(x_2(t+1))$ $= f_3 \circ f_2(x_5(t))$
4	$f_4(x_5(t))$	$f_4(x_5(t+1))$ $= f_4 \circ f_5(x_0(t))$	$f_4(x_5(t))$
5	$f_5(x_0(t))$	$f_5(x_0(t))$	$f_5(x_0(t))$

Fig. 1: Above: interaction graphs associated to the three different update schedules considered in example 2.1. Below: a table giving the dependencies between states of nodes according to the update schedule of the network.

A network $N = (G, \mathcal{F})$, updated according to a block-sequential update schedule s is denoted by $N(s)$. Its dynamics is defined by the following *global transition function*:

$$F_s : \begin{cases} \{0,1\}^n & \to & \{0,1\}^n \\ x & \mapsto & (f_0^s(x), \ldots, f_{n-1}^s(x)) \end{cases} \tag{1}$$

where $\forall i \in V,\ f_i^s$ is the *local transition function of node* i *relative to* s and is defined by:

$$f_i^s(x) = f_i(x^{(s,i)}), \qquad \forall j \in V,\ x_j^{(s,i)} = \begin{cases} x_j & \text{if } s(j) \geq s(i) \\ f_j^s(x) & \text{if } s(j) < s(i). \end{cases} \tag{2}$$

In particular, if $s = \pi$ then $\forall i \in V,\ f_i^s = f_i$ and the global transition function simplifies to: $F^{(\pi)}(x) = (f_0(x), \ldots, f_{n-1}(x))$. When there is no ambiguity as to what network is being considered, for any initial configuration $x \in \{0,1\}^n$, we write $x = x^s(0)$ and $x^s(t) = F_s^t(x)$ (where F_s is composed t times) and when there is no ambiguity either on s, we write $x = x(0)$ and $x(t) = F_s^t(x)$. With this notation, (1) and (2) mean that $\forall i, j \in V$ such that $(j,i) \in A$, $x_i(t+1)$ depends on $x_j(t)$ if $s(j) \geq s(i)$, and on $x_j(t+1)$ if $s(j) < s(i)$.

For a network N updated with a particular update schedule s, we define a new interaction graph $G^s = (V, A^s)$, the interaction graph relative to s (see figure 1) such that $A^s = \{(j,i) \mid x_i^s(t+1)$ depends on $x_j^s(t)\}$. By an easy induction, this set of arcs can be shown to be equal to:

$$A^s = \{(j,i) \mid \text{there exists in } G \text{ a directed path } \{v_0 = j, v_1 \ldots, v_l = i\} \text{ from } j \text{ to } i \text{ such that } s(j) \geq s(v_1) \text{ and } \forall\ 1 \leq k < l,\ s(v_k) < s(v_{k+1})\}. \quad (3)$$

An important point is that when $s = \pi$, $G^\pi = G$. Further, define $N^s = (G^s, \mathcal{F}^s)$ to be the network whose interaction graph is G^s and whose set of local transition functions is $\mathcal{F}^s = \{f_i^s \mid i \in V\}$. Then, as one may check, the dynamics of $N^s(\pi)$ is identical to that of $N(s)$: the global transition functions of both networks are equal to F_s. As a result, provided a characterisation of the graphs G^s, we may bring our study of networks updated with arbitrary block-sequential update schedules back to the study of networks updated in parallel.

The dynamics of a network N updated with an update schedule s is described by its *iteration graph* $\mathcal{I}(N(s))$ (and also, from the previous paragraph by the iteration graph $\mathcal{I}(N^s(\pi))$) whose nodes are the configurations of N and whose arcs are the transitions $(x(t), x(t+1))$ from one configuration to another. Since the set of configurations of any finite sized network is finite, all trajectories necessarily end up looping, *i.e.,* $\forall\ x(0) \in \{0,1\}^n,\ \exists t, p,\ x(t+p) = x(t)$. *Attractors* of $N(s)$ are orbits of such configurations $x(t)$ for which there exists a $p \in \mathbb{N}$ such that $x(t) = x(t+p)$. The smallest such p is called the period of the attractor. Attractors of period one are called fixed points.

3 Boolean automata circuits

As mentioned in the introduction, we pay special attention here to a particular instance of Boolean automata networks called *Boolean automata circuits* [7]. A *circuit* of size n is a digraph denoted by $\mathbb{C}_n = (V, A)$. Its set of nodes $V = \{0, \ldots, n-1\}$ is identified with $\mathbb{Z}/n\mathbb{Z}$ so that, considering two nodes i and j, $i + j$ designates the node $i + j \mod n$. The set of arcs of $\mathbb{C}_n$ is $A = \{(i, i+1) \mid i \in \mathbb{Z}/n\mathbb{Z}\}$. A *Boolean automata circuit* is a Boolean automata network whose interaction graph is a circuit. Since any node i in this graph has a unique incoming neighbour, $i-1$, its local transition function f_i is either equal to the identity function $id : a \in \{0,1\} \mapsto a$ or to the negation function $neg : a \in \{0,1\} \mapsto \neg a = 1 - a$. In the first case, the arc $(i-1, i)$ is said to be *positive* and in the second case it is said to be *negative*. When there is an even number of negative arcs in the circuit, then the the *sign* of the (Boolean automata) circuit is said to be positive. Otherwise it is said to be negative.

Let $C = (\mathbb{C}_n, \mathcal{F})$ be a Boolean automata circuit of size n whose set of local transition functions is $\mathcal{F} = \{f_i \mid i \in \mathbb{Z}/n\mathbb{Z}\})$. Let s be an arbitrary block-sequential update schedule of C. For any node $i \in V$, let us note:

$$i^* \ = \ max\{k < i \mid s(k) \geq s(k+1)\}.$$

where the maximum is taken cyclically so that the number of arcs on a path from i^* to i is is minimal. From (3), it holds that $A^s = \{(i^*, i) \mid i \in V\}$ and it can be shown that $\forall i \in \mathbb{Z}/n\mathbb{Z},\ f_i^s = F[i, i^* + 1]$ where:

$$\forall i, j \in V,\ F[j,i] = \begin{cases} f_j \circ f_{j-1} \circ \ldots \circ f_i & \text{if } i \leq j \\ f_j \circ f_{j-1} \circ \ldots \circ f_0 \circ f_{n-1} \circ \ldots \circ f_i & \text{if } j < i \end{cases}$$

Following the remarks made in the previous section, the dynamics of $C(s)$ is identical to that of $C^s(\pi) = (\mathbb{C}_n^s, \mathcal{F}^s)$ where $\mathcal{F}^s = \{F[i, i^* + 1] \mid i \in \mathbb{Z}/n\mathbb{Z}\}$. Let us describe the digraph $\mathbb{C}_n^s$. To do this, we first define the *inversions* of C relative to s:

$$inv(s) = A \setminus A^s = \{(i, i+1) \mid s(i) < s(i+1)\}.$$

For nodes of an inversion $(i, i+1)$, $x_{i+1}^s(t+1)$ depends on $x_i^s(t+1)$ instead of $x_i^s(t)$ as is the case when $s(i+1) \leq s(i)$ and when, in particular, $s = \pi$. Obviously, the number of inversions is strictly smaller than n. The only block-sequential update schedule that has no inversions is the parallel update schedule π. From the characterisation of A^s given in equation 3, we derive that the nodes i^* (*i.e.*, the nodes $i \in \mathbb{Z}/n\mathbb{Z},\ \exists j \in \mathbb{Z}/n\mathbb{Z},\ i = j^*$) form a circuit in $\mathbb{C}_n^s$ of size $n - |inv(s)|$. The $|inv(s)|$ other nodes that do not belong to this circuit depend on one and only one node in it (as in Figure 2). And since the composition of all functions f_i is necessarily equal to the composition of all functions $F[i, i^* + 1]$, the sign of this circuit is equal to the sign of the original circuit $\mathbb{C}_n$. From this description of the network C^s, we may now derive the following result:

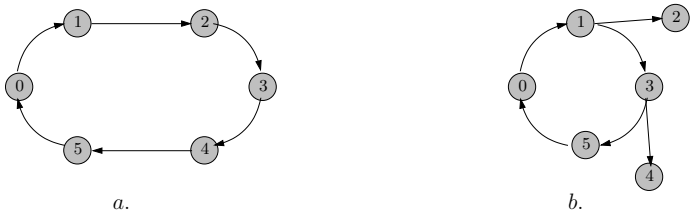


Fig. 2: a. The underlying interaction graph $\mathbb{C}_6$ of a network $C = (\mathbb{C}_6, \mathcal{F})$. b. The interaction graph of C^s where $s \equiv (2)(3,4)(0,1,5)$ and $inv(s) = \{(2,3),(4,5)\}$. The underlying circuit of size 4 in this second interaction graph has as set of nodes $\{0,1,3,5\} = \{i \in \mathbb{Z}/6\mathbb{Z} \mid \exists j \in \mathbb{Z}/6\mathbb{Z},\ i = j^*\}$.

Proposition 3.1 *Let $C = (\mathbb{C}_n, \mathcal{F})$ be a Boolean automata circuit of size n and let s and r be two block-sequential update schedules of C. Then:*

(i) The dynamics induced by s, that of $C(s)$, and the dynamics induced by r, that of $C(r)$, are identical if and only if $inv(s) = inv(r)$.

(ii) If $inv(s) \neq inv(r)$, then the dynamics induced by s and by r have no attractor of period $p > 1$ in common.

(iii) If $|inv(s)| = k$, then for any $p \in \mathbb{N}$, $C(s)$ has as many attractors of period p than any Boolean automata circuit of size $n - k$, of same sign as C and updated with the parallel update schedule.

Proof: *(i)* follows directly from theorem 1 of [5] and *(iii)* is derived from the description of the structure of $\mathbb{C}_n^s$ made in the previous paragraph. To prove *(ii)*, suppose that $(i, i+1) \in inv(r) \setminus inv(s)$ and that there exists $x = x^s(t) = x^r(t) \in \{0,1\}^n$ such that $x^s(t+1) = x^r(t+1)$. Then:

$$x^s_{i+1}(t+2) = f_{i+1}(x^s_i(t+2)) = F[i+1, i^*+1](x^s_{i^*}(t+1))$$

and

$$x^r_{i+1}(t+2) = f_{i+1}(x^r_i(t+1)) = f_{i+1}(x^s_i(t+1)) = F[i+1, i^*+1](x^s_{i^*}(t))$$

where $i^* = max\{k < i \mid s(k) \geq s(k+1)\}$ (as above). By the injectivity of $F[i+1, i^*+1]$, this implies that if $x^s(t+2) = x^r(t+2)$ then $x_{i^*}(t+1) = x_{i^*}(t)$. Now, if x belongs to an attractor that is induced identically by both s and r, then $\forall t \in \mathbb{N},\ x^s(t) = x^r(t)$. As result, in this case, $\forall t \in \mathbb{N},\ x^s_{i^*}(t+1) = x^r_{i^*}(t) = x^s_{i^*}(t)$ (*i.e.,* the state of node i^* is fixed in the attractor). As one can check this leads to states of all nodes being fixed in the attractor which therfore is a fixed point. □

In relation with point *(ii)* of Proposition 3.1 above, recall that if the dynamics of a network has fixed points for a certain update schedule, then it has the same fixed points for every other update schedule. The important consequence of point *(iii)* of Proposition 3.1 is that from the results in [7] concerning the number of attractors of Boolean automata circuits updated in parallel, we may derive the number of attractors of each period and in total of any Boolean automata circuit updated with any block-sequential update schedule:

Corollary 3.1 *Let $C = (\mathbb{C}_n, \mathcal{F})$ be a Boolean automata circuit of size n and s a block-sequential update schedule of C such that $|inv(s)| = k$:*

- *If C is positive, then the total number of attractors in the dynamics of $C(s)$ is given by $\mathtt{T}_p^+$ below. For any integer p, the number of attractors of period p is either 0 if p does not divide $n-k$ or it is $\mathtt{A}_p^+$:*

$$\mathtt{T}_p^+ = \frac{1}{n-k} \cdot \sum_{p \mid n-k} \psi(\frac{n-k}{p}) \cdot 2^p, \qquad \mathtt{A}_p^+ = \frac{1}{p} \cdot \sum_{d \mid p} \mu(\frac{p}{d}) \cdot 2^d.$$

- *If C is negative, then the total number of attractors in the dynamics of $C(s)$ is given by $\mathtt{T}_p^-$ below. For any integer p, the number of attractors of period p is either 0 if $n-k$ cannot be written $n-k = q \times \frac{p}{2}$ where $q \in \mathbb{N}$ is odd, or it is $\mathtt{A}_p^-$:*

$$\mathtt{T}_p^- = \frac{1}{2n} \cdot \sum_{odd\ p \mid n} \psi(\frac{n}{p}) \cdot 2^p, \qquad \mathtt{A}_p^- = \frac{1}{p} \cdot \sum_{odd\ d \mid \frac{p}{2}} \mu(d) \cdot 2^{\frac{p}{2d}}.$$

Above, μ is the Möbius (see [9, 1]) function and ψ the Euler totient function.

Following Proposition 3.1, we define the equivalence relation between update schedules that relates r and s if and only if $inv(s) = inv(r)$. $[s]$ denotes the equivalence class of s for this relation. Proposition 3.2 below sums up some results concerning this relation:

Proposition 3.2 *Let $C = (\mathbb{C}_n, \mathcal{F})$ be a Boolean automata circuit of size n.*

(i) The total number of distinct dynamics induced by the different update schedules of C is $\sum_{k=0}^{n-1} \binom{n}{k} = 2^n - 1$.

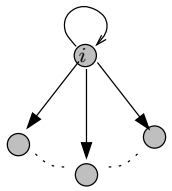


Fig. 3: Interaction graph relative to one of the n equivalence classes of update schedules that have $n-1$ inversions. Each one of these classes is characterised by the unique node $i \in \mathbb{Z}/n\mathbb{Z}$ that is such that $(i, i+1)$ is *not* an inversion and contains exactly one update schedule which is sequential, namely, the update schedule $s_i \equiv (i+1)(i+2)\dots(i-1)(i)$ such that $inv(s_i) = \{(j, j+1),\ j \neq i\}$. Because there is a loop over node i in this graph, the dynamics of $C(s_i)$ contains only fixed points if $\mathbb{C}_n$ is positive and only attractors of period 2 if $\mathbb{C}_n$ is negative.

(ii) In every class $[s]$, $s \neq \pi$, there exists a sequential update schedule. Given the set of inversions of the class, a sequential update schedule can be constructed effectively in $\mathcal{O}(n)$ steps.

(iii) Given a set of $p > 1$ configurations of C, $\mathcal{A} = \{x(0), \dots, x(p-1)\}$, we can determine in $\mathcal{O}(p \cdot n)$ steps whether there exists a block-sequential update schedule s such that $C(s)$ has $\mathcal{A}$ as an attractor of period p. If such an update schedule exists, with Algorithm 5 below, in $\mathcal{O}(p \cdot n)$ steps, we can compute its set of inversions as well as a sequential update schedule inducing the same dynamics.

Algorithm 1: Finding a sequential update schedule that induces a particular attractor of a given Boolean automata circuit if it exists

Input: $C = (\mathbb{C}_n, \mathcal{F})$ and $\mathcal{A} = \{x(0), \dots, x(p-1)\}$.

begin

1. In $\mathcal{O}(p \cdot n)$ steps, compute the set $\mathcal{A}^\pi = \{y(t) = F_\pi(x(t-1)) \mid 0 \leq t < p\}$;
2. In $\mathcal{O}(p \cdot n)$, compute the set $inv = \{(i-1, i) \mid \exists t \leq p,\ x_i(t) \neq y_i(t)\}$;
3. In $\mathcal{O}(n)$ steps, compute a sequential update schedule s using the set inv;
4. In $\mathcal{O}(p \cdot n)$ steps, compute the set $\mathcal{A}_s = \{F_s^t(x(0)) = x^s(t) \mid 0 \leq t < p\}$ and check that $\mathcal{A}_s = \mathcal{A}$. If not, then no update schedule induces $\mathcal{A}$ as an attractor;
5. Otherwise, output s.

Proof: Point *(i)* of Proposition 3.2 above is a direct consequence of points *(i)* and *(ii)* of Proposition 3.1 and of the fact that the number of distinct equivalence classes of update schedules with k inversions is $\binom{n}{k}$ (*i.e.,* the number of different sets of k inversions).

To prove Point *(ii)*, let us show that for every set of $k < n$ inversions, there exists a sequential update schedule s that satisfies exactly these k inversions. Thus, let inv be a set of $|inv| = k$ inversions and let $G = (V, A)$ be the acyclic digraph whose set of nodes is that of $\mathbb{C}_n$ (*i.e.,* $V = \{0, \dots, n-1\}$) and whose set of arcs is $A = \{(i, i+1) \notin inv\} \cup \{(i+1, i) \mid (i, i+1) \in inv\}$ (in other words, G is obtained by inverting all arcs of $\mathbb{C}_n$ that belong to inv). Then, any sequential update schedule s whose set of inversions is inv satisfies the following:

$$\forall (i,j) \in A,\ s(i) > s(i)$$

so that such a sequential update schedule s can be obtained in linear time using a topological ordering algorithm on digraph G.

Finally, to prove Point *(iii)* and Algorithm 5, suppose that s is an existing block-sequential update schedule that induces $\mathcal{A}$ as an attractor, *i.e.,* $\forall t < p,\ x^s(t+1) = f_i^s(x(t)) = x(t+1)$. Let us show that its set of inversions $inv(s)$ is necessarily equal to inv. Suppose that $(i-1, i) \notin inv(s)$. Then, $\forall t < p,\ x_i(t+1) = x_i^s(t+1) = f_i^s(x(t)) = f_i(x_{i-1}(t)) = y_i(t+1)$ and consequently, $(i-1, i) \notin inv$. Now, $\forall i \in \mathbb{Z}/n\mathbb{Z}$, again, let $i^* = max\{j < i,\ (i^*, i^*+1) \notin inv(s)\}$. It is easy to prove that the state of any node j such that $\exists i,\ j = i^*$ necessarily changes in all attractors induced by s and in particular in $\mathcal{A}$. Suppose that $(i-1, i) \in inv(s)$. Let $T < p$ be such that $x_{i^*}(T) \neq x_{i^*}(T+1)$. Then, the following holds:

$$x_i(T+2) = x_i^s(T+2) = F[i, i^*+1](x_{i^*}(T+1)) \text{ and}$$
$$y_i(T+2) = f_i(x_{i-1}(T+1)) = f_i(x_{i-1}^s(T+1)) = f_i \circ F[i-1, i^*+1](x_{i^*}(T))$$

so that $x_i(T+2) \neq y_i(T+2)$ and consequently $(i-1, i) \in inv$. □

4 Conclusion

Following the work presented in this paper, we believe that most combinatoric problems concerning the dynamics Boolean automata circuits updated with block-sequential update schedules have now been dealt with. We know the exact value of both the total number of attractors and the number of attractors of period p, $\forall p \in \mathbb{N}$, in the dynamics of positive and negative Boolean automata circuits of any size updated with the synchronous, sequential and the block-sequential update schedules. We also know how many different dynamics can be induced by the set of block-sequential update schedules of a Boolean automata circuit.

One important question, however, remains unanswered: "What are the sizes of the equivalence classes of block-sequential update schedules that yield the same dynamics?". For the very particular cases of $[\pi]$ and of the classes of update schedules with $n-1$ inversions (where n is the size of the circuit) we know that the size of the classes is 1. We also obtained a very intricate formula for the size of classes of update schedules having *consecutive* inversions only. It implies that the sizes of such classes is exponential as may certainly be that of many other classes. One motive (amongst others) for studying this question follows from A. Elena's work. In his PhD thesis [8], Elena computed statistics of the number of attractors of threshold Boolean automata networks as well as of their periods averaging over all networks (of sizes between 3 and 6) and all update schedules. For both, he found particularly small values. Now, as we have already mentioned, it is known that underlying circuits play an important role in the dynamics of

a network with an arbitrary structure. Knowing the answer to this question would help us to understand better the averages found by Elena.

Therefore, beyond this question, we believe that there are two obvious extensions needed of our combinatoric analysis of the dynamics of circuits: one towards more general networks, that is, networks with arbitrary underlying interaction graphs. In line, with [4], this would need to relate the dynamics of arbitrary networks with that of there embedded circuits. The second extension needed is in the direction of other update schedules. Although understanding the dynamics of networks under block-sequential update schedules is a first notable step, these update schedules remain rather unadapted to the modelisation of biological networks. One may indeed argue that it is rather unrealistic that a network updates infallibly every one of its nodes exactly once and according to the exact same order at every time step. It seems more likely, that, on the contrary, some nodes may be updated more often than others and that the updating of nodes may depend on some parameters in a way that cannot be translated by giving an *order* of update as do block-sequential update schedules.

Acknowledgements

We thank the Basal project-CMM and the Fondecyt 1100003.

References

[1] T. M. Apostol. *Introduction to analytic number theory*. Springer-Verlag, 1976.

[2] J. Aracena, J. Demongeot, and E. Goles. Fixed points and maximal independent sets in and-or networks. *Discrete Applied Mathematics*, 138:277–288, 2004.

[3] J. Aracena, J. Demongeot, and E. Goles. On limit cycles of monotone functions with symmetric connection graph. *Theoretical Computer Science*, 322:237–244, 2004.

[4] J. Aracena, J. Demongeot, and E. Goles. Positive and negative circuits in discrete neural networks. *IEEE Transactions on Neural Networks*, 15:77–83, 2004.

[5] J. Aracena, E. Goles, A. Moreira, and L. Salinas. On the robustness of update schedules in boolean networks. *Biosystems*, 97, 2009.

[6] J. Demongeot, A. Elena, and S. Sené. Robustness in regulatory networks: a multi-disciplinary approach. *Acta Biotheoretica*, 56(1-2):27–49, 2008.

[7] J. Demongeot, M. Noual, and S. Sené. On the number of attractors of boolean automata circuits. WAINA, Perth, Australia, 2010. IEEE Press. To appear.

[8] A. Elena. *Robustesse des réseaux d'automates booléens à seuil aux modes d'itération. Application à la modélisation des réseaux de régulation génétique*. PhD thesis, Université Joseph Fourier - Grenoble, 2009.

[9] C. F. Gauss and A. A. (tr.) Clarke. *Disquisitiones Arithemeticae*. Yale University Press, 1965.

[10] D. Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8:437–479, 1902.

[11] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Journal of Mathematical Biology*, 5:115–133, 1943.

[12] M. Noual. On the dynamics of two particular classes of boolean automata networks: Boolean automata circuits and or networks. Technical report, TIMC-IMAG (Grenoble) and CMM (Santiago de Chile), 2009.

[13] A. Richard and J.P. Comet. Necessary conditions for multistationarity in discrete dynamical systems. *Discrete Applied Mathematics*, 155(18):2403–2413, 2007.

[14] F. Robert. *Discrete Iterations*. Springer-Verlag, 1986.

[15] R. Thomas. On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations. *Springer Series in Synergetics*, 9:180–193, 1981.

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 55–74

The fractal structure of cellular automata on abelian groups

Johannes Gütschow[1], Vincent Nesme[1], and Reinhard F. Werner[1]

[1] *Institut für Theoretische Physik, Universität Hannover, Appelstraße 2, 30167 Hannover*

It is a well-known fact that the spacetime diagrams of some cellular automata have a fractal structure: for instance Pascal's triangle modulo 2 generates a Sierpinski triangle. Explaining the fractal structure of the spacetime diagrams of cellular automata is a much explored topic, but virtually all of the results revolve around a special class of automata, whose main features include irreversibility, an alphabet with a ring structure and a rule respecting this structure, and a property known as being (weakly) p-Fermat. The class of automata that we study in this article fulfills none of these properties. Their cell structure is weaker and they are far from being p-Fermat, even weakly. However, they do produce fractal spacetime diagrams, and we will explain why and how.

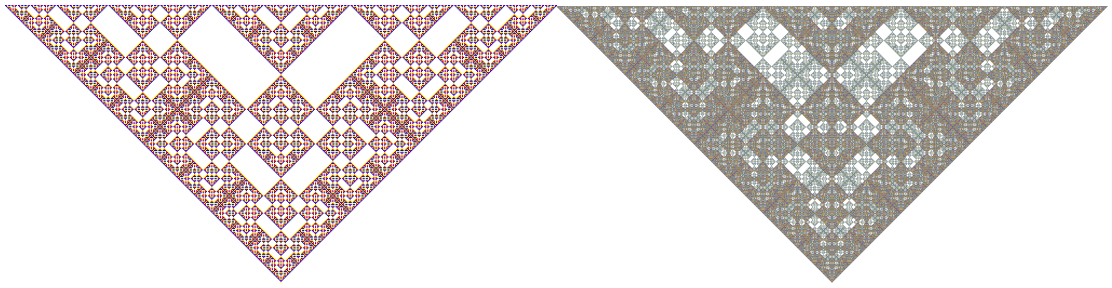
These automata emerge naturally from the field of quantum cellular automata, as they include the classical equivalent of the Clifford quantum cellular automata, which have been studied by the quantum community for several reasons. They are a basic building block of a universal model of quantum computation, and they can be used to generate highly entangled states, which are a primary resource for measurement-based models of quantum computing.

Keywords: fractal, abelian group, linear cellular automaton, substitution system

Introduction

The fractal structure of cellular automata (CA) has been a topic of interest for several decades. In many works on linear CA, the authors present ways to calculate the fractal dimension or to predict the state of an arbitrary cell at an arbitrary time step, with much lower complexity than by running the CA step by step; however, their notions of linearity are quite different. Often only CA that use states in $\mathbb{Z}_p$[(i)] are studied; other approaches are more general, but still make certain assumptions on the time evolution or the underlying structure of the CA. In this work we try to loosen these restrictions as far as possible. We consider one-dimensional linear CA whose alphabet is an abelian group. We show how they can be described by $n \times n$ matrices with polynomial entries and use this description to derive a recursion relation for the iterations of the CA. This recursion relation enables us to formulate the evolution of the spacetime diagram as a

(i) We use the simple notation $\mathbb{Z}_d$ for the cyclic group of order d, instead of $\mathbb{Z}/d\mathbb{Z}$, as we are concerned with finite groups only.



(a) Time evolution of Θ with scalars in $\mathbb{Z}_2$.

(b) Time evolution of Θ with scalars extended to $\mathbb{Z}_4$.

Figure 1: (a) is a projection of (b) induced by $\mathbb{Z}_4^2 \twoheadrightarrow \mathbb{Z}_2^2$.

matrix substitution system, which in turn gives us the means to calculate the fractal dimension of the spacetime diagram.

Our interest in the fractal structure of CA on abelian groups stems from our study of Clifford quantum cellular automata (CQCA) [SVW08]. We first noticed the self similar structure while studying their long time behaviour [GUWZ10, Güt10]. A CQCA maps Pauli matrices to tensor products of Pauli matrices times a phase. If we neglect the phase, we can identify the Pauli matrices X, Y, Z with the elements of $\mathbb{Z}_2^2$ via the mapping $X \mapsto \binom{1}{0}$, $Y \mapsto \binom{1}{1}$, $Z \mapsto \binom{0}{1}$, $\mathbb{1} \mapsto \binom{0}{0}$. Using this mapping we can simulate CQCA with linear CA on the alphabet $\mathbb{Z}_2^2$. The CA corresponding to CQCA have to fulfill rather strong conditions: they have to be reversible and preserve a symplectic form which encodes the commutation relations of the Pauli matrices [SVW08]. While our analysis is now much more general, our main example Θ, whose spacetime diagram is shown in figure 1a, is the classical part of a CQCA.

Our paper is organized as follows: in section 1 we give our definition of a linear cellular automaton, introduce the formalism we will be working with, and state the main result: every linear cellular automaton has a fractal structure. We also introduce the example Θ which will be the focal point of this article. In section 2, we give an intuitive idea as to why the spacetime diagram of Θ exhibits a fractal structure. We then proceed, in section 3, to expose an algorithm taking as input the local transition rule and outputting a description of the spacetime diagram. This allows us to compute salient features of these fractals, such as their fractal dimension and their average colour.

1 Definitions

1.1 Generalities on summable automata

1.1.1 Monoids

We want to discuss "summable automata", for which it makes sense to talk about the influence of a single cell on every other cell, and where the global transition function can be reconstructed by "summing" all these influences. So, if Σ denotes the alphabet, instead of the usual local transition function $\Sigma^I \to \Sigma$, a summable automaton is naturally defined by a function $\Sigma \to \Sigma^I$. What

is then the minimal structure on Σ that would make such a definition work? These influences have to be "summed", so we need an operation on Σ. Since the strip is infinite, an infinitary operation would do, but that wouldn't give us much to work with. Instead, it seems reasonable to consider a binary operation $+$. In the same spirit, when we think of the superposition of influences coming from each cell, no notion of order between the cells is involved; even if in the one-dimensional case a natural order can be put on the cells, it would be less than clear what to do in higher dimensions. We require therefore that $+$ be associative and commutative. The last requirement comes from the fact that, given only the global transition function, we want to be able to isolate the influence of one cell; that is why we demand that $+$ have an identity element, which makes now $(\Sigma, +)$ an abelian monoid. Of course, in order for all of this to be relevant, the transition function has to be a morphism.

Let I be some finite subset of $\mathbb{Z}$ and f a morphism from Σ to Σ^I. From f one can define the global transition function as an endomorphism F of $\Sigma^{\mathbb{Z}}$ by

$$F : \begin{pmatrix} \Sigma^{\mathbb{Z}} & \to & \Sigma^{\mathbb{Z}} \\ r = (r_n)_{n\in\mathbb{Z}} & \mapsto & \left(\sum_{i\in I} f(r_{n-i})_i\right)_{n\in\mathbb{Z}} \end{pmatrix}. \tag{1}$$

Let σ be the right shift on $\Sigma_{\mathbb{Z}}$, i.e. $\sigma(r)_n = r_{n-1}$. We have $F \circ \sigma = \sigma \circ F$, which means F is translation invariant. Also, $F(r)_n$ depends only on the values r_{n-i} for $i \in I$; since I is finite, F is a one-dimensional cellular automaton on the alphabet Σ, with neighbourhood included in $-I$. Conversely, if F is an endomorphism of $\Sigma^{\mathbb{Z}}$ defining a cellular automaton over the alphabet Σ, then one can choose a neighbourhood I, and define, for $i \in I$,

$$f(s)_i = F(\bar{s})_i, \tag{2}$$

where $\bar{s}$ is the word of $\Sigma_{\mathbb{Z}}$ defined by $\bar{s}_n = \begin{cases} s & \text{if } n = 0 \\ e & \text{otherwise} \end{cases}$, e denoting the neutral element of Σ.

1.1.2 Groups

We will now consider the case when Σ is a (finite abelian) group. For p prime, let Σ_p be the subgroup of Σ of elements of order a power of p; then Σ is isomorphic to $\prod_p \Sigma_p$, and every endomorphism of $\Sigma^{\mathbb{Z}}$ factorises into a product of endomorphisms of the $\Sigma_p^{\mathbb{Z}}$'s. It is therefore enough to study the case of the (abelian) p-groups: let us assume Σ is a p-group.

It is a well-known fact (see for instance section I-8 of [Lan93]) that Σ is isomorphic to $\mathbb{Z}_{p^{k_1}} \times \mathbb{Z}_{p^{k_2}} \times \cdots \times \mathbb{Z}_{p^{k_d}}$ with $k_d \geq k_{d-1} \geq \ldots \geq k_1 = k$. Consider an endomorphism α of Σ and let e_j denote $(0, \ldots, 0, 1, 0, \ldots, 0)$, where the 1 lies in position j. When $i \geq j$, there is a natural embedding $s_{i,j}$ of $\mathbb{Z}_{p^{k_i}}$ into $\mathbb{Z}_{p^{k_j}}$, namely the multiplication by $p^{k_j - k_i} \in \mathbb{N}$. Since e_j has order p^{k_j}, $\alpha(e_j)_i \in \mathbb{Z}_{p^{k_i}}$ has to be in the image of $s_{j,i}$ when $i \leq j$. We can therefore associate to α the endomorphism of $\mathbb{Z}_{p^k}^d$ given by the matrix $A(\alpha) \in \mathscr{M}_d(\mathbb{Z}_{p^k})$ defined by $A(\alpha)_{i,j} = p^{k_j - k_i} \alpha\left(e_j\right)_i$.

For instance, if G is $\mathbb{Z}_{32} \times \mathbb{Z}_4 \times \mathbb{Z}_2$, and α is defined by $\alpha(1,0,0) = (3,2,1)$, $\alpha(0,1,0) = (24,0,1)$ and $\alpha(0,0,1) = (16,2,0)$, then the corresponding matrix of $\mathscr{M}_3(\mathbb{Z}_{32})$ would be

$$A(\alpha) = \begin{pmatrix} 3 & 3 & 1 \\ 16 & 0 & 1 \\ 16 & 2 & 0 \end{pmatrix}.$$

Let us give a summary of the construction we have just exposed.

Proposition 1 *For every finite abelian p-group G and endomorphism α of G, there are positive integers k and d, an embedding s of G into $\mathbb{Z}_{p^k}^d$, and an endomorphism $A(\alpha)$ of $\mathbb{Z}_{p^k}^d$ such that the following diagram commutes:*

$$\begin{array}{ccc} G & \xrightarrow{\alpha} & G \\ \downarrow{\scriptstyle s} & & \downarrow{\scriptstyle s} \\ \mathbb{Z}_{p^k}^d & \xrightarrow{A(\alpha)} & \mathbb{Z}_{p^k}^d \end{array} \tag{3}$$

This implies that to study the behaviour of CA on abelian groups, it is enough to study the case where these groups are of the form $\mathbb{Z}_{p^k}^d$.

1.1.3 R-modules

We will actually consider the more general case where R is a finite commutative ring, and Σ is a free R-module of dimension d, i.e. isomorphic to R^d. The first reason for doing so is that it does not complicate the mathematics. It will also appear more efficient to understand, for instance, $\mathbb{F}_{2^4}$ as a 1-dimensional vector space over itself than as a 4-dimensional vector space over $\mathbb{F}_2$: the former simply bears more information, and therefore implies more restrictions on the form of a CA, so that more can be deduced.

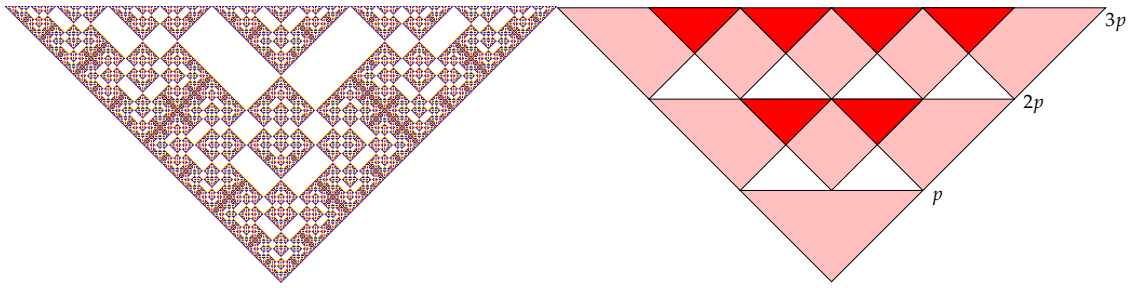
For any ring B, $B\left[u,u^{-1}\right]$ denotes the ring of Laurent polynomials over B; it is the ring of linear combinations of integer powers (negative as well as nonnegative) of the unknown u. Applying this to $B = \mathrm{Hom}_R(\Sigma)$, we can associate to the function f the Laurent polynomial $\tau(f) \in \mathrm{Hom}_R(\Sigma)\left[u,u^{-1}\right]$ defined by

$$\tau(f) = \sum_{n\in\mathbb{Z}} f(\cdot)_n u^n. \tag{4}$$

τ is an isomorphism of R-algebras between the linear cellular automata on the alphabet Σ with internal composition rules $(+,\circ)$ and $\mathrm{Hom}_R(\Sigma)\left[u,u^{-1}\right]$, which can be identified with $\mathscr{M}_d\left(R\left[u,u^{-1}\right]\right)$ because $\Sigma \simeq R^d$; we are going to think and work in this former algebra, so from now on a linear cellular automaton $T = \tau(f)$ will be for us an element of $\mathscr{M}_d\left(R\left[u,u^{-1}\right]\right)$.

1.2 Related work

Many papers have been published about the fractal structure of cellular automata spacetime diagrams. We give here a short review and point out the differences to our approach. When we mention d and k we are referring to $\mathscr{M}_d\left(\mathbb{Z}_k[u,u^{-1}]\right)$.



(a) Time evolution of Θ starting from $\xi = \binom{1}{0}$.

(b) Time evolution of a general nearest neighbour p-Fermat CA.

Figure 2: This figure shows that Θ cannot be a p-Fermat CA. In a p-Fermat CA at least the white areas are filled by the neutral element e; Θ has a different pattern.

[Wil87] In this work, Willson considers the case $d = 1$, $k = 2$. In order to determine the fractal dimension of the spacetime diagram, he analyses how blocks of length n in the configuration of time step t are mapped to such blocks in steps $2t$ and $2t + 1$, a technique we also use in section 3.

[Tak90, Tak92] Takahashi generalises Willson's work to the case $d = 1$, with no restriction on the value of k.

[HPS93, HPS01] Haeseler et al. study the fractal time evolution of CA with special scaling properties, the weakest of them being "weakly p-Fermat", where p is some integer, which includes the case $d = 1$, $k = p$. Let us briefly introduce the p-Fermat property and show why the CA studied by us do not have to be p-Fermat. Let π_p be the scaling map

$$\pi_p(\xi)_x = \begin{cases} \xi_y & \text{if } x = py \\ e & \text{otherwise} \end{cases} . \tag{5}$$

A CA T is weakly p-Fermat if for all $s \in \Sigma$, $n \in \mathbb{N}$ and $x \in \mathbb{Z}$, $T^{np}(\bar{s})_x = e \Leftrightarrow \pi_p T^n(\bar{s})_x = e$.

Let us now consider

$$\Theta = \begin{pmatrix} 0 & 1 \\ 1 & u^{-1} + 1 + u \end{pmatrix} \in \mathscr{M}_2(\mathbb{Z}_2[u, u^{-1}]). \tag{6}$$

We will use this example throughout the paper. It generates the time evolution depicted in Figure 2a. A general nearest-neighbour p-Fermat CA produces a time evolution that reproduces itself after p steps in at most three copies located at positions $\{-p; 0; p\}$. After $2p$ steps we have five copies at most. This creates areas filled with the neutral element e shared by all p-Fermat CA for a fixed p. In figures 2a and 2b we can easily see that Θ does not exhibit these areas. Thus it is not p-Fermat. Furthermore p-Fermat CA that are not periodic are irreversible, while we also allow reversible CA, Θ being again one example.

[AHPS96, AHP+97] Allouche et al. study recurrences in the spacetime diagram of linear cellular automata, from the angle of k-automatic sequences, which we will not define in this paper. However they require Σ to be an abelian ring and the CA to be a ring homomorphism, which is again essentially the case $d = 1$.

[Moo97, Moo98] Moore studies CA with an alphabet A on a staggered spacetime, where every cell c is only influenced by two cells a and b of the last time step. The update rule is $c = a \bullet b$. He requires $(A, \bullet)$ to be a quasigroup and studies different special cases. First let us note that these CA are irreversible, while ours don't have to be. Thus, although it is possible to bring our CA in the form of a staggered CA, the results of Moore do not apply. In his setting, our CA would be of the form $c = a \bullet b = f(a) + g(b)$ for some homomorphisms f and g. For $(A, \bullet)$ to be a quasigroup means

$$\forall a, b \in A \; \exists! x, y \in A \quad a \bullet x = b \wedge y \bullet x = b. \tag{7}$$

In our case, the required equalities translate respectively as $g(x) = b - f(a)$ and $f(y) = b - g(a)$. The right-hand sides are each arbitrary elements of A, thus f and g have to be isomorphisms, as indeed required in [Moo97].

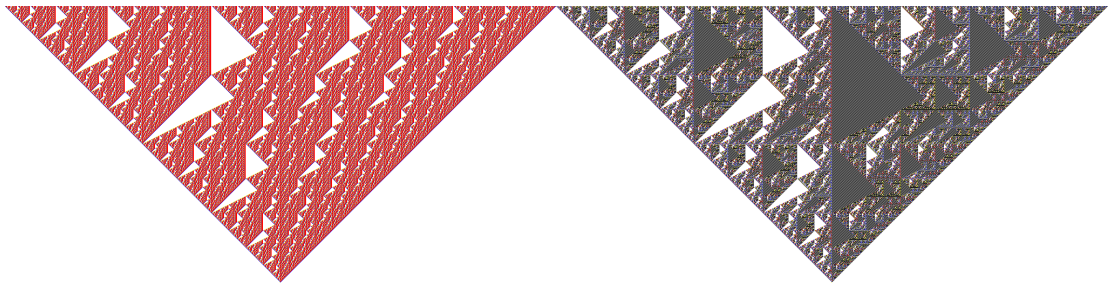
The angle of study of Moore is also different: he does not exactly study fractal properties of the CA, but rather the complexity of the prediction — "What will be the state of this cell after t steps?". Describing the spacetime diagram with a matrix substitution system is an alternative way of proving that prediction is an easy task — for instance it makes it **NC**.

[Mac04] Macfarlane uses Willson's approach and generalises parts of it to matrix-valued CA, his examples including Θ. However, the transition matrix is obtained heuristically — "by scrutiny of figure 9" — from the spacetime diagram, instead of being algorithmically derived from the transition rule (as in the present work). The conclusion (section 6) suggests that the analysis of Θ is easily generalisable to matrices of various sizes over various rings, so in a sense the present article is but an elaboration of the concluding remark of [Mac04], although we have to say we do not find this generalisation to be that obvious.

The heart of the proof is in section 3. In a nutshell, whereas most of the techniques used in our article can be traced back to older articles, the new one that allows us to extend the analysis to a larger class of automata is the introduction of α in equation (20). The idea in doing so is to get rid of the complicated noncommutative ring structure and go back to a simple linear recurrence, as state in Proposition 4. Since a linear recurrence is precisely where the analysis started from, it could seem at first sight that nothing is gained in the process, but the new recurrence actually does not define a cellular automaton. Instead of defining line $n + 1$ from line n, it cuts right through to line mn, thus establishing a scaling property.

1.3 Different CA

While we use Θ, which has very special properties (being reversible, over a field of characteristic two, and described by a 2×2 matrix) as our example throughout the paper, the analysis applies of course to all other linear CA. In this section we give a short overview over the variety of



(a) Time evolution of Θ_u.

(b) Time evolution of $T_{\mathbb{F}_4}$.

Figure 3: Spacetime diagrams of non-clifford CAs.

spacetime diagrams these CA generate. Let us start by small changes to Θ. For our first example we keep $m = k = 2$, but change the determinant to u. We only change one entry of the matrix:

$$\Theta_u = \begin{pmatrix} 0 & u \\ 1 & u^{-1} + 1 + u \end{pmatrix}. \tag{8}$$

The spacetime diagram is displayed in figure 3a and shows how much difference a small change in the update rule can make for the spacetime diagram.

Let us now modify Θ in a more subtle fashion:

$$\Theta_{k=4} = \begin{pmatrix} 0 & 1 \\ 1 & u^{-1} + 1 + u \end{pmatrix}. \tag{9}$$

The hidden difference with Θ is the underlying ring, which has now been extended from $\mathbb{Z}_2$ to $\mathbb{Z}_4$. $\Theta_{k=4}$ contains in some sense more information than Θ, since Θ is induced from $\Theta_{k=4}$ by the projection $\mathbb{Z}_4 \twoheadrightarrow \mathbb{Z}_2$. Consequently, the spacetime diagram of Θ is nothing but a projection of that of $\Theta_{k=4}$, as illustrated in figure 1b.

The last CA we want to present lies in $\mathscr{M}_2\left(\mathbb{F}_4[u, u^{-1}]\right)$, where $\mathbb{F}_4$ is the finite field of order 4, here identified with $\mathbb{F}_2[\omega]/(\omega^2 + \omega + 1)$. The corresponding matrix is

$$T_{\mathbb{F}_4} = \begin{pmatrix} 0 & \omega \\ u^{-1} & (\omega + 1)u^{-1} + \omega + u \end{pmatrix}. \tag{10}$$

If one wants to avoid calculations in $\mathbb{F}_4$, this CA can be translated to a CA in $\mathscr{M}_4\left(\mathbb{Z}_2[u, u^{-1}]\right)$, namely

$$\tilde{T}_{\mathbb{F}_4} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ u^{-1} & 0 & u^{-1} + u & u^{-1} + 1 \\ 0 & u^{-1} & u^{-1} + 1 & 1 + u \end{pmatrix}.$$

Its spacetime diagram, shown in figure 3b, contains patches of checkerboard pattern. Somehow, they trivialise most of the usual properties of the figure: for instance they make its fractal dimension 2, even if the fractal structure can hardly be considered trivial. In order to access

more interesting properties, it is possible to blank this pattern out, considering it as just "another shade of white". This can be trivially done on the matrix substitution system, by removing the states from which the blank state is inaccessible.

1.4 Coloured spacetime diagrams

The mainstream setting when studying the fractal structure of spacetime diagrams is monochromatic; we introduce colours in the picture.

Instead of considering simple compact subsets of the plane, we will have a finite set of colours $\mathscr{C}$ and compact subsets of $\left(\mathbb{R}^2\right)^{\mathscr{C}}$. Let $b \notin \mathscr{C}$ be the additional "blank" colour and $\mathfrak{c} : \Sigma \to \mathscr{C} \cup \{b\}$ a colouring of Σ such that $\mathfrak{c}(0) = b$. To determine a *coloured spacetime diagram*, we need furthermore to be given an automaton $T \in \mathscr{M}_d\left(R\left[u, u^{-1}\right]\right)$, an initial state $\xi \in R^d$, and an integer n. The corresponding coloured spacetime diagram is then the rescaled diagram obtained by iteratively applying T n times on ξ.

Formally, for $n, i, j \in \mathbb{N}$, let $S_{n,i,j}$ be the full square centred in $\frac{1}{n}(i,j)$ and whose edges, parallel to the axes, are of length $\frac{1}{n}$. To each positive integer n and colour $c \in \mathscr{C}$ is associated a compact subset of the plane $\mathscr{P}_n(c)$ which is the union of the $S_{n,i,j}$'s such that $0 \leq j \leq n$ and $\mathfrak{c}\left(T^j\left(\xi\right)_i\right) = c$. The coloured spacetime diagram of order n is then the function $\mathscr{P}_n : c \mapsto \mathscr{P}_n(c)$. A sequence of coloured patterns $\left(\mathscr{P}_n\right)_{n\in\mathbb{N}}$ of spacetime diagrams is said to converge to some coloured pattern $\mathscr{P}_\infty$ if for every $c \in \mathscr{C}$, $\left(\mathscr{P}_n(c)\right)_{n\in\mathbb{N}}$ converges to $\mathscr{P}_\infty(c)$ for the Hausdorff distance.

We can now state our main result.

Theorem 1 *Let G be a finite abelian p-group. For every cellular automaton over G that is also a group homomorphism, there exists a positive integer m such that for every fixed initial state the coloured spacetime diagrams of order p^{mn} converge when n goes to infinity.*

In general, to know about the fractal structure of a cellular automaton over some finite group G, write G as a product of p-groups and study each p-component of the spacetime diagram independently; according to Theorem 1, each component generates a fractal pattern. Then, since the logarithms of the prime numbers are rationally independent, it is possible to find a sequence of resized spacetime diagrams that converges towards a superposition of these different components with arbitrary independent rescaling coefficients, but there is no direct generalisation of the theorem. For instance, even in the simple case of Pascal's triangle modulo 6, there is no real number $\alpha > 0$ such that the diagrams of order $\lfloor \alpha^n \rfloor$ converge; however those of order t_n will converge as soon as the fractional parts of $\log_3(t_n)$ and $\log_2(t_n)$ both converge, and then their limits determine the limit pattern. The situation is described very briefly in the section 5 of [Tak92].

1.4.1 Matrix substitution systems

We will show how to find a suitable description of the limit pattern in the rest of this article. We now explain exactly what it means to generate a coloured picture by rules of substitution, and how to take the limit of all these pictures. This is a generalisation of the usual monochromatic description that can be found for instance in [MGAP85, Wil87, HPS93], and which corresponds in our setting to the case where all the colours are mapped to "black".

Let V be a finite alphabet; because we want colours, compared with the usual definition of a matrix substitution system, we don't have to include a special "empty" letter. A matrix substitution system is then a function $\mathscr{D} : V \to V^{[\![1;r]\!]^2}$; for some integer r. Together with a set of colours $\mathscr{C}$ and a colouring $\mathfrak{c} : V \to \mathscr{C}$, it defines coloured patterns, much in the same way cellular automata do. With the previous notations, at each step n, the pattern $\mathscr{P}_n$ is the union of squares $S_{r^n,i,j}$ of different colours, for different i's and j's; each one of them is indexed by some letter in V.

Then at step $n+1$, each coloured square of colour c indexed by $v \in V$ present in the nth step pattern is replaced by r^2 smaller squares that pave it; these smaller squares are given by $\mathscr{D}(v)$ and indexed accordingly. To such a matrix substitution system we can associate a multigraph $\Gamma = (V, E)$ where the set of vertices is V and we put as many edges from v to w as there are w's in $\mathscr{D}(v)$.

A *plain* matrix substitution system is one of the usual kind: no colouring, and V contains a special letter ε such that $\mathscr{D}(\varepsilon)$ is a matrix full of ε's and $\mathfrak{c}(\varepsilon) = b$. In the multigraph associated to a plain matrix substitution system, ε is excluded from the set of vertices.

We want to generalise the usually property of convergence of the patterns defined by plain matrix substitution systems. This will be done by the conjunction of the two following propositions. Let us first remind some notions on graphs: the *period* of a graph is the greatest common divisor of the lengths of all the cycles in Γ; a graph is *aperiodic* if it has period 1.

Proposition 2 *If every strongly connected component of Γ is aperiodic, then $(\mathscr{P}_n)_{n\in\mathbb{N}}$ converges.*

Proof: To each colour $c \in \mathscr{C}$ we associate the plain matrix substitution system $\mathscr{D}^c$, obtained from $\mathscr{D}$ simply by turning some letters into ε. For $v \in V$, let $X_c(v)$ be the set of integers n such that there exists a path of length n in Γ connecting v to a letter of the colour c. Since the strongly connected component containing v is aperiodic, $X_c(v)$ is either finite or cofinite. Those letters $v \in V$ such that $X_c(v)$ is finite are sent to ε, and this defines $\mathscr{D}^c$. If $X_c(v)$ is finite and v' can be reached from v, then $X_c(v')$ is also finite; therefore, $\mathscr{D}^c$ is indeed a substitution system. Let M be such that for every $v \in V$, either $X_c(v)$ or its complement is strictly bounded by M.

Let us now compare two sequences of figures. The first one is $(\mathscr{P}_n(c))$, the subpattern of colour c defined by $\mathscr{D}$. The second one is $(\mathscr{P}^c_n)$, the one obtained from $\mathscr{D}^c$; we know that it converges to some compact $\mathscr{D}^c_\infty$. By construction, $\mathscr{P}_{n+M}(c)$ is included in $\mathscr{P}^c_n$, and for every black square of $\mathscr{P}^c_n$, there is a black subsquare in $\mathscr{P}_{n+M}(c)$. The Hausdorff distance between $\mathscr{P}_{n+M}(c)$ and $\mathscr{P}^c_n$ therefore converges to 0, so that $(\mathscr{P}_n(c))$ converges to $\mathscr{D}^c_\infty$. □

For a graph Γ, let $\Gamma^k = (V, E^k)$ be defined by

$$(v_0, v_k) \in E^k \iff (\exists v_1, \ldots, v_{k-1} \in V \quad \forall i \in \{0, \ldots, k-1\} \quad (v_i, v_{i+1}) \in E)\,. \tag{11}$$

Proposition 3 *For every (multi)graph Γ, there exists k such that every strongly connected component of Γ^k is aperiodic.*

Proof: Each strongly connected component Δ of Γ has a period $p(\Delta)$, so that $\Delta^{p(\Delta)}$ is aperiodic. Let k_0 be the least common divisor of the $p(\Delta)$'s; then each strongly connected component of Γ induces an aperiodic graph in Γ^{k_0}, but it is possible that, in the process, it broke down into several connected components, so that Γ^{k_0} might not have the required property. The procedure then has to be repeated from Γ^{k_0} to obtain $\Gamma^{k_0 k_1}$, and so on. Since the strongly connected components of $\Gamma^{k_0 \cdots k_{i+1}}$ are included in those of $\Gamma^{k_0 \cdots k_i}$, this process reaches a fixed point, which is a graph with the required property. □

Ergo, a coloured matrix substitution system defines a convergent coloured pattern when considering the steps that are a multiple of some well-chosen integer m. So, in order to prove Theorem 1, all we need to do is find such a substitution system. This will be done in a special case in the next section, and in the general case in section 3.

2 A special recursion scheme for Θ

The aim of this section is to give the most direct and natural explanation of the fractal structure generated by Θ that we are aware of. Modulo some caveat, it applies effortlessly to all invertible elements T of $\mathscr{M}_2\left(R[u,u^{-1}]\right)$, where R is a finite abelian ring of characteristic 2. This section is not vital to the proof of the general case presented in 3, and can therefore be skipped by the impatient reader.

We will deduce informally the basic structure of the spacetime diagrams from a simple recursion relation for the 2^n-th powers of T. The characteristic polynomial of T, $P_T(X)$, is equal to $X^2 + (\operatorname{tr} T) + \det T$. According to Cayley-Hamilton theorem, $P_T(T) = 0$, so $T^2 + (\operatorname{tr} T)\, T + (\det T)\, \mathbb{I} = 0$. Multiplying this equation by T^{-1}, we get $T = (\det T)\, T^{-1} + (\operatorname{tr} T)\, \mathbb{I}$. Let us denote $\widetilde{T} = (\det T)\, T^{-1}$, $\det T = u^{\varepsilon}$, which we will name the *dual* of T; since we are in characteristic 2, by repeatedly taking the square of this equality, we obtain

$$\forall n \in \mathbb{N} \quad T^{2^n} = \widetilde{T}^{2^n} + (\operatorname{tr} T)^{2^n}\, \mathbb{I}. \tag{12}$$

Taking the trace of this equation, we get $\operatorname{tr} T^{2^n} = \operatorname{tr} \widetilde{T}^{2^n}$; in particular, $\operatorname{tr} T = \operatorname{tr} \widetilde{T}$ so Equation (12) is also valid when swapping T and $\widetilde{T}$. Let I_T be a finite set, and the λ_i's elements of R such that $\operatorname{tr} T = \sum_{i \in I_T} \lambda_i u^i$. Then we have

$$\forall n \in \mathbb{N} \quad (\operatorname{tr} T)^{2^n} = \sum_{i \in I_T} (\lambda_i)^{2^n}\, u^{2^n i}. \tag{13}$$

We do not yet specify the initial state; as a matter of fact, it will prove to be largely irrelevant. The only thing we ask for now is that it is nontrivial (and finite).

Consider for instance Θ; we have $\det \Theta = 1 = u^{\varepsilon=0}$ and $\lambda_i = \chi_{\{-1;0;1\}}(i)$. We start with the spacetime diagram corresponding to 2^n steps; it is rescaled to a triangle with vertex coordinates $\{(0,0),(-1,1),(1,1)\}$. Taking equations (12) and (13), we can see that the state at the 2^n-th time step can be decomposed into a sum of several copies of the initial state (the positions are governed by the coefficients of the trace) and a configuration that can be derived by applying $\widetilde{T}^{2^n}$ to the initial state. In the next 2^n steps, this configuration will contract itself to the initial

state, which is shifted according to ε as $\widetilde{T}$ is the inverse of T composed with the shift $u^{\varepsilon}\mathbb{I}$. The copies of the original initial state evolve according to T. This is illustrated in Figure 4. The figure

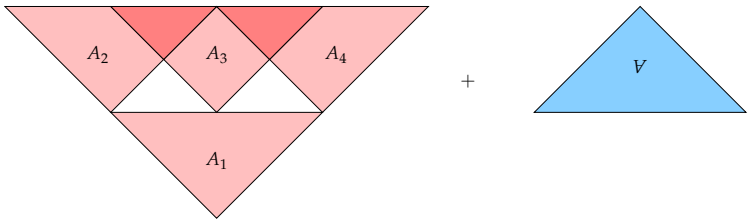
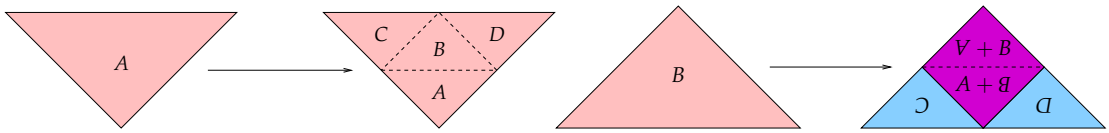


Figure 4: The whole figure is the sum of $|I| + 2$ parts

suggests to divide the spacetime diagram into four parts A, B, C, and D as shown in Figure 5a which overlap only on a single cell strip at the borders.



(a) The first substitution rule.

(b) The second substitution rule.

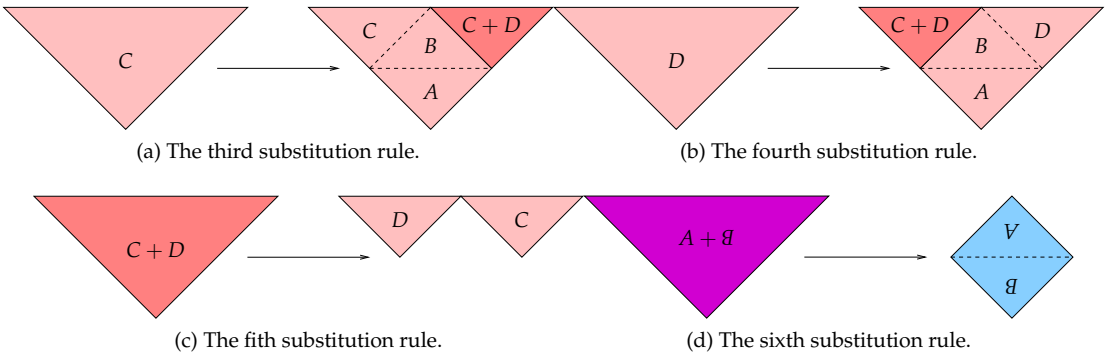
Figure 5: The first and the second substitution rules.

A_2, A_3, ad A_4 are copies of A_1, and V is marked by an upside down A because it is the reverse evolution of the initial state under the CA $\widetilde{\Theta}$ — so actually it should logically be named $\widetilde{V}$ or $\underset{\sim}{V}$, but since it always appears upside-down while A always appears straight on its feet, there is no risk of confusion.

Let us assume that the sequence of rescaled spacetime diagrams up to step 2^n actually converges. Then that means A_1 should be, in the limit, a copy of the whole picture A, downsized by a factor 2, so we rename it A. This gives us the first substitution rule, represented in Figure 5a. The other three parts are still unknown, and we will name these patterns B, C and D. Equation 12 tells us what the other substitution rules are.

Since Equation 12 remains true after swapping T and $\widetilde{T}$, V admits likewise a partition into V, β, $\mathcal{O}$ and $\mathcal{C}$. Summing all the parts shown in Figure 4, we get the top rightmost pattern of Figure 7. Superimposing our first substitution rule (Figure 5a) with our second step of the decomposition, we get the new substitution rules represented by Figures 5b, 6a and 6b.

All the other substitution rules are deducible from these ones. First they are linear: for instance the substitutions for $C + D$ is the sum of the substitutions for C and D, as shown in Figure 6c. We don't know *a priori* what the sum of two patterns is, but we know that summing a pattern with itself should give 0, for we are working in characteristic 2. In this case $A + A$ and $B + B$ cancel out. In figure 7, one can see how the characteristic white spaces emerge from the above substitution rule.



(a) The third substitution rule.

(b) The fourth substitution rule.

(c) The fith substitution rule.

(d) The sixth substitution rule.

Figure 6: The third and the fourth substitution rules.

The problem with this scheme is that what is happening goes beyond simple juxtaposition of patterns. There can be cancellation at the border between patterns. And, sure, we know $C + C$ is blank, but how do we know $C + D$, for instance, is not? Well, generally we don't. If the initial state is itself blank, then the whole figure would be, and all tiles being blank is certainly a fixed point for all the substitution rules.

In this case, however, everything turns out well. It should first be noticed that in every part of the picture not tagged as "blank", an A pattern can be found by refining a few more steps. Formally, let $G = (V, E)$ be the graph whose vertices are the different possible tiles (i.e., in this case, A, B, C, D, V, $\mathcal{A}$, $\mathcal{D}$, $\mathcal{C}$, and the sums modulo 2 of tiles having compatible shapes, including the blank tile 0), and edges represent the transition rule in the following way: each vertex has four edges coming out of it, each one pointing to one of its subtiles. In our case, the graph has the property that the set of vertices accessible from A, minus 0, form a strongly connected component.

We may then distinguish two cases: either A has a point in its interior, or it has points only on its border triangle. In the first case, the unique non-empty compact defined by the substitution rule is actually the figure we're looking for. Indeed, it follows from the property of connexity — cf. Proposition 2 — that every non-zero tile actually appearing in the decomposition of the figure has a non-empty interior. Thus, no matter what happens at the boundary between tiles, the figure constructed this way will always converge to the same compact.

3 Recursion and matrix substitution system

We will now present a general method to calculate the fractal dimensions and average colour of the spacetime diagrams of linear CA in $\mathscr{M}_d(\mathbb{Z}_{p^l}[u, u^{-1}])$. We will again demonstrate the method using our example Θ, whereas the derivation is carried out for the general case. Thus the algorithm works as well on all the CA obeying our definition, e.g. the CA presented in section 1.3, as it works on Θ. Of course with larger neighbourhoods and groups of higher order the substitution system becomes larger and larger, so that one might want to use a computer to derive the substitution system.

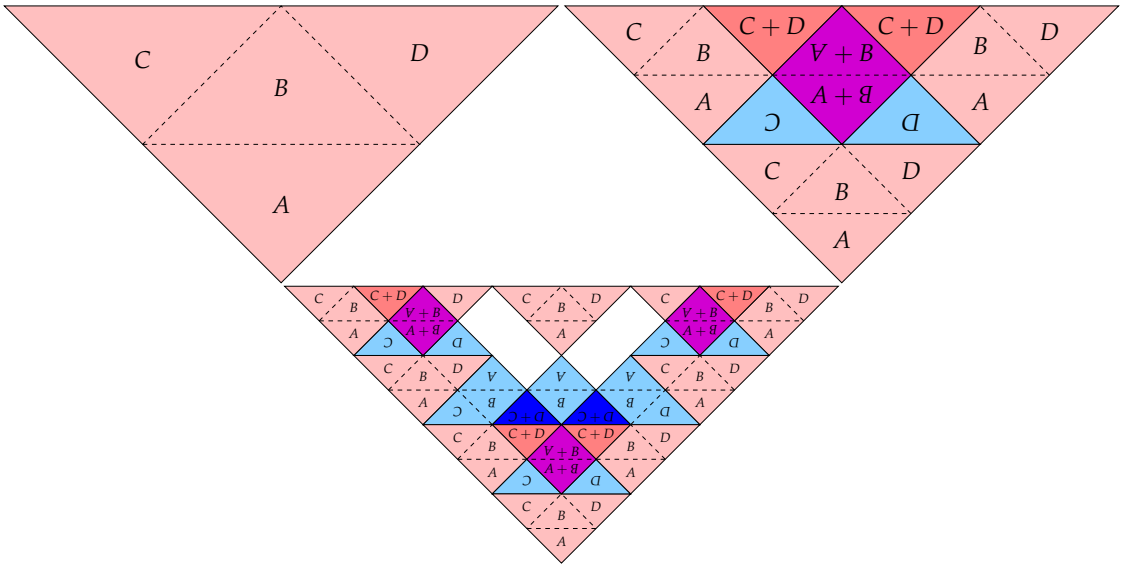


Figure 7: Three decomposition steps for the spacetime diagram of Θ. One can clearly see the characteristic white spaces emerging.

Our approach is the following: from the minimal polynomial Π of the CA T (or any other polynomial fulfilling $\Pi(T) = 0$) we derive a recursion relation for the T_x^y's, the coefficients in u^x of T^y. We then forget about every other piece of information we might have on T, to concentrate only on this recursion: this shows that the fractal structure, except for contingent blank spaces, can be essentially derived just from the minimal polynomial of T. We further develop our recursion scheme for T until we can express every T_x^y in terms of the T_i^j of the first m time steps with coefficients $\alpha_j(x-i, y)$. With a simple grouping of cells we deduce a matrix substitution system that enables us to generate the spacetime diagram of step $t = k^{n+1}$ directly from step $t = k^n$. Using this substitution system we can calculate the fractal dimension, the average colouring, and given an initial state also the whole space time diagram.

Now let $\Pi(X) \in R[u, u^{-1}][X]$ be a monic polynomial such that $\Pi(T) = 0$:

$$\Pi(X) = X^m - \sum_{j=0}^{m-1} \lambda_{\Pi,j} X^j. \tag{14}$$

According to the Cayley-Hamilton theorem, which we can apply in our case because T is an endomorphism of a finite-dimensional free module over an abelian ring (see Theorem 3.1. of [Lan93]), the characteristic polynomial of T fulfills this condition, therefore we can always find such a polynomial. Let $\mathcal{I}$ be the finite set of exponents i's such that the coefficient in u^i of Π, seen as an element of $R[X][u, u^{-1}]$, is nonzero, so that we can write $\lambda_{\Pi,j} = \sum_{i \in \mathcal{I}} \lambda_{\Pi,i,j} u^i$. $\mathcal{I}$ is not to be confused with the neighbourhood of the CA, I, which won't play any role from now on.

For any $x, y, n \in \mathbb{N}$, $(x+y)^p \equiv x^p + y^p [p]$, and if $x \equiv y[p^n]$ then $x^p \equiv y^p [p^{n+1}]$. Therefore, for any $x, y \in R$ and $n \in \mathbb{N}$, $(x+y)^{p^{n+l-1}} = (x^{p^n} + y^{p^n})^{p^{l-1}}$. Since the powers of T commute pairwise, we get

$$T^{p^{n+2(l-1)}m} = \left(\sum_{j=0}^{m-1} \lambda_{\Pi,j}^{p^{n+l-1}} T^{p^{n+l-1}j} \right)^{p^{l-1}} = \left(\sum_{j=0}^{m-1} \left(\sum_{i \in \mathcal{I}} \lambda_{\Pi,i,j}^{p^n} u^{p^n i} \right)^{p^{l-1}} T^{p^{n+l-1}j} \right)^{p^{l-1}}. \tag{15}$$

For each i, j, the sequence $\lambda_{\Pi,i,j}^{p^n}$ is ultimately periodic. There exists therefore integers N and M such that for all i, j, $\lambda_{\Pi,i,j}^{p^{M+N}} = \lambda_{\Pi,i,j}^{p^M}$. Let $k = p^N$; substituting n by $M + Nn$ in (15), we get

$$T^{k^n p^{M+2(l-1)}m} = \left(\sum_{j=0}^{m-1} \left(\sum_{i \in \mathcal{I}} \lambda_{\Pi,i,j}^{p^M} u^{k^n p^M i} \right)^{p^{l-1}} T^{k^n p^{M+l-1}j} \right)^{p^{l-1}}. \tag{16}$$

Hence, if we note $m' = p^{M+2(l-1)}m$ and expand this equation, we find that there is some finite subset $\mathcal{I}'$ of $\mathbb{Z}$ and some elements $\mu_{i,j}$ of R, for $i \in \mathcal{I}'$ and $j \in [\![0; m'-1]\!]$, such that for all $n \in \mathbb{N}$,

$$T^{k^n m'} = \sum_{j=0}^{m'-1} \sum_{i \in \mathcal{I}'} \mu_{i,j} u^{k^n i} T^{k^n j}. \tag{17}$$

We have now used everything we needed to know from the multiplicative structure on the ring of matrices. As announced at the end of section 1.2, we will now get rid of it and concentrate only on the linear recurrence relation that we have just derived. Remember that $T^j \in \mathscr{M}_d\left(R\left[u, u^{-1}\right]\right)$, and we are interested in the coefficient of T^j in u_i, denoted T_i^j, so that $T^j = \sum_{i \in \mathcal{I}} T_i^j u^i$. We thus get the following relation: $T_x^{k^n m' + y} = \sum_{i \in \mathcal{I}'} \sum_{j=0}^{m'-1} \mu_{i,j} T_{x-k^n i}^{y+k^n j}$, which we rewrite in this form:

$$T_x^y = \sum_{(i,j) \in \mathcal{I}' \times [\![0; m'-1]\!]} \mu_{i,j} T_{x+f_{i,j}(y)}^{g_{i,j}(y)} \tag{18}$$

where $f_{i,j}(y) = -k^n i$ and $g_{i,j}(y) = y - k^n(m' - j)$, which of course works with any n, but we will choose $n = \left\lfloor \log_k \frac{y}{m'} \right\rfloor$. In order to emphasise that the rest of the proof will use only a minimal structure, we state in the next proposition what will actually be proven, and change the notation from T, which was an element of $\mathscr{M}_d\left(R[u, u^{-1}]\right)$, to Ξ, an element of a more arbitrary R-module. It is straightforward to check that T fulfils the hypotheses of the proposition.

Proposition 4 *Let M be a finite R-module, k a positive integer, Λ a finite set of indices, and for $i \in \Lambda$, $\mu_i \in R$, $f_i : [\![m; +\infty[\![\to \mathbb{Z}$ and $g_i : [\![m; +\infty[\![\to \mathbb{N}$ such that for all $y \in [\![m; +\infty[\![$ and $t \in [\![0; k-1]\!]$,*

- $g_i(y) < y$;

- $f_i(ky+t) = kf_i(y)$ *and* $g_i(ky+t) = kg_i(y)+t$.

For $x \in \mathbb{Z}\times\mathbb{N}$, *let* $\Xi_x^y \in M$ *be such that when* $y \geq m$,

$$\Xi_x^y = \sum_{i\in\Lambda} \mu_i \Xi_{x+f_i(y)}^{g_i(y)}. \tag{19}$$

Then there exists a finite set E *and a function* $e : \mathbb{Z}\times\mathbb{N} \to E$ *such that*

- Ξ_x^y *is a function of* $e(x,y)$;
- *for* $s,t \in [\![0;k-1]\!]$, $e\left(kx+s,ky+t\right)$ *is a function of* s, t, *and* $e(x,y)$.

The introduction of a new function e in this proposition comes from the need of a scaling property, expressing that the state at point $(kx+s,ky+t)$ can be deduced from the state at point (x,y). Such a property does not follow immediately from Equation (19), but it is possible to expand the state space from M to E, and to put more information into than e than into Ξ, so as to fulfill the scaling property. An immediate consequence of this proposition is that the spacetime diagrams of Ξ_x^y of order k^n can be described by coloured matrix substitution systems, so that Theorem 1 will follow from Proposition 3. Let us now prove Proposition 4.

If $y \geq m$, we can apply Equation (19) with a unique n to give an expression of Ξ_x^y in terms of a linear combination of $\Xi_{x'}^{y'}$'s with $y' < y$. Starting from any point $(x,y) \in \mathbb{Z}\times\mathbb{N}$ and performing these operations recursively we get to the expression

$$\Xi_x^y = \sum_{i\in\mathbb{Z}} \sum_{j=0}^{m-1} \alpha_{i,j}(x,y)\Xi_i^j \tag{20}$$

which we take as a definition of $\alpha_{i,j}(x,y)$. Since the relation (19) is invariant under translations of the parameter x, we have $\alpha_{i,j}(x,y) = \alpha_{0,j}(x-i,y)$. Noting $\alpha_j := \alpha_{0,j}$, we get the following equation:

$$\Xi_x^y = \sum_{i\in\mathbb{Z}} \sum_{j=0}^{m-1} \alpha_j(x-i,y)\Xi_i^j. \tag{21}$$

Let us now show that $\alpha_\cdot(kx+s,ky+t)$, for $s,t \in [\![0;k-1]\!]$, is a function of s, t, and the $\alpha_\cdot(x',y)$'s, where x' ranges over some neighbourhood of x. By substituting x with $kx+s$ and y with $ky+t$ in Equation (19), we get

$$\Xi_{kx+s}^{ky+t} = \sum_{i\in\Lambda} \mu_i \Xi_{k(x+f_i(y))+s}^{kg_i(y)+t}. \tag{22}$$

Because the indices and exponents of Ξ on the left and right side of this equation have undergone the same transformation $(x,y) \mapsto (kx+s,ky+t)$, we arrive recursively at this point

$$\Xi_{kx+s}^{ky+t} = \sum_{i\in\mathbb{Z}} \sum_{j=0}^{m-1} \alpha_j(x-i,y)\Xi_{ki+s}^{kj+t} \tag{23}$$

which we want to compare to the following equation, directly deduced from (21):

$$\Xi^{ky+t}_{kx+s} = \sum_{i\in\mathbb{Z}} \sum_{j=0}^{m-1} \alpha_j(kx+s-i, ky+t)\Xi^j_i \tag{24}$$

Of course, there can be terms in (23) with $kj+t \geq m$, so that the decomposition is not over: it then needs to be performed to its end. What we could have wished for would have been for $\alpha_\cdot(kx+s, ky+t)$ to depend only on $\alpha_\cdot(x,y)$. This is not quite true; instead the final decomposition of (23) relates the coefficients $\alpha_j(kx+s-i, ky+t)$ of the Ξ^j_i in (24) to sums of the $\alpha_j(x-i,y)$'s. Therefore $\alpha_\cdot(kx+s, ky+t)$ depends on the $\alpha_\cdot(x+i,y)$'s for i ranging over some set $\mathscr{D}$. However, this not much of a problem, as a simple grouping will take care of it — a technique commonly attributed to [Wil87]. Let us show that $\mathscr{D}$ is finite. Since $j \in [\![0; m-1]\!]$ and $t \in [\![0; k-1]\!]$, the $kj+t$ appearing as an exponent of Ξ in (23) is in $[\![0; km-1]\!]$. We therefore have to use at most $(k-1)\,m$ recursive calls to (19) in order to get down to coefficients Ξ^y_x with $y < m$, each one of them decreasing the exponent by at least 1. Each one of them also increases the index by $f_i(y)$; since both Λ and $[\![0; km-1]\!]$ are finite, the set of possible $f_i(y)$'s is also bounded by some M, and the total variation in the index, i.e. $\mathscr{D}$, is then bounded by $(k-1)\,mM$; let us say $\mathscr{D} \subseteq [\![d_{\min}; d_{\max}]\!]$.

Let us now introduce $\beta_{\Pi,\cdot}(x,y) = (\alpha_\cdot(x-i,y))_{i\in\mathscr{D}'}$, where $\mathscr{D}' = [\![\delta_{\min}; \delta_{\max}]\!]$, $\delta_{\max}$ being such that $\delta_{\max} \geq d_{\max} + \left\lceil \frac{\delta_{\max}}{k} \right\rceil$ and $\delta_{\min}$ such that $\delta_{\min} \leq d_{\min} - 1 + \left\lceil \frac{\delta_{\min}+1}{k} \right\rceil$. This time, for $s,t \in \{0;\ldots;k-1\}$, $\beta_{\Pi,\cdot}(kx+s, ky+t)$ does really depend only on $\beta_{\Pi,\cdot}(x,y)$. Indeed, $\beta_{\Pi,\cdot}(kx+s, ky+t) = (\alpha_\cdot(kx+s-i, ky+t))_{i\in\mathscr{D}'}$, and each $\alpha_\cdot(kx+s-i, ky+t)$ depends only on $\left(\alpha_\cdot\left(x + \left\lfloor \frac{s-i}{k} \right\rfloor - j, y\right)\right)_{j\in\mathscr{D}}$; the choice of $\mathscr{D}'$ has been made so that $j - \left\lfloor \frac{s-i}{k} \right\rfloor \in \mathscr{D}'$. This concludes the proof of Proposition 4, since we can choose $E = M^{[\![0;m-1]\!]\times\mathscr{D}'}$, with $e(x,y)(j,i) = \alpha_j(x-i,y)$.

3.1 Example: Θ

In the case of Θ, Equation (23) becomes

$$\Xi^{2y+t}_{2x+s} = \sum_i \alpha_{\Theta,0}(x-i,y)\Xi^t_{2i+s} + \alpha_{\Theta,1}(x-i,y)\Xi^{2+t}_{2i+s}. \tag{25}$$

The first term is now elementary, but the second one has to be decomposed once more, i.e.

$$\Xi^{2+t}_{2i+s} = \Xi^t_{2i+s} + \Xi^{1+t}_{2i+s-1} + \Xi^{1+t}_{2i+s} + \Xi^{1+t}_{2i+s+1}, \tag{26}$$

which is the end of it if $t = 0$, but not if $t = 1$, where we get

$$\Xi^3_{2i+s} = \Xi^0_{2i+s-1} + \Xi^0_{2i+s} + \Xi^0_{2i+s+1} + \Xi^1_{2i+s-2} + \Xi^1_{2i+s+2}. \tag{27}$$

The substitution rule of $\alpha_{\Theta,\cdot}$ can then be written in the following way, where for convenience $\alpha_{\Theta,\cdot}$ is represented as $\boxed{\begin{matrix}\alpha_{\Theta,1}\\ \alpha_{\Theta,0}\end{matrix}}$:

$$\boxed{\alpha_{\Theta,\cdot}(x,y)} \rightarrow \begin{array}{|c|c|}\hline \alpha_{\Theta,\cdot}(2x,2y+1) & \alpha_{\Theta,\cdot}(2x+1,2y+1)\\ \hline \alpha_{\Theta,\cdot}(2x,2y) & \alpha_{\Theta,\cdot}(2x+1,2y)\\ \hline \end{array}$$

$$= \begin{array}{|c|c|}\hline \alpha_{\Theta,0}(x,y)+\alpha_{\Theta,1}(x-1,y)+\alpha_{\Theta,1}(x+1,y) & 0\\ \alpha_{\Theta,1}(x,y) & \alpha_{\Theta,1}(x,y)+\alpha_{\Theta,1}(x+1,y)\\ \hline \alpha_{\Theta,1}(x,y) & \alpha_{\Theta,1}(x,y)+\alpha_{\Theta,1}(x+1,y)\\ \alpha_{\Theta,0}(x,y)+\alpha_{\Theta,1}(x,y) & 0\\ \hline \end{array}$$

If we follow exactly what has been said in the general case, we ought to consider the grouping $\{-2;\cdots;3\}$. However, this general bound is obviously too rough in the case of Θ, where we will just have to take $\{-1;2\}$. We will represent the grouping in the form

$$\begin{array}{|cccc|}\hline \alpha_{\Theta,1}(x-1,y) & \alpha_{\Theta,1}(x,y) & \alpha_{\Theta,1}(x+1,y) & \alpha_{\Theta,1}(x+2,y)\\ \alpha_{\Theta,0}(x-1,y) & \alpha_{\Theta,0}(x,y) & \alpha_{\Theta,0}(x+1,y) & \alpha_{\Theta,0}(x+2,y)\\ \hline \end{array}.$$

The alphabet has thus size 256, and the substitution system is described by

$$\begin{array}{|cccc|}\hline a & b & c & d\\ e & f & g & h\\ \hline \end{array} \rightarrow \begin{array}{|cccc|cccc|}\hline 0 & a+c+f & 0 & b+d+g & a+c+f & 0 & b+d+g & 0\\ a+b & b & b+c & c & b & b+c & c & c+d\\ \hline a+b & b & b+c & c & b & b+c & c & c+d\\ 0 & b+f & 0 & c+g & b+f & 0 & c+g & 0\\ \hline \end{array}$$

Let us denote by A the matrix having a 1 in position a and 0 elsewhere, B the matrix having a 1 only in position b, and so on. For these matrices, we will denote the sum of matrices by a simple juxtaposition: AB will mean $A+B$, as the matrix multiplication has no meaning in this context.

Since $T^0_x = \delta_{x0}T^0_0$, the starting position, with which we describe the whole line number 0, is $\begin{array}{|c|c|c|c|c|c|c|}\hline \cdots & 0 & H & G & F & E & 0 & \cdots\\ \hline \end{array}$. Since we have, for instance, the rule $\boxed{F} \rightarrow \begin{array}{|c|c|}\hline B & A\\ \hline F & E\\ \hline \end{array}$, the graph derived from this substitution system is aperiodic; that means that, in whatever way $A, B, C, \ldots$ are represented, either as coloured dots or as white dots, the pattern converges, and the fractal structure is described by this matrix substitution system (see Section 1.4).

To calculate the fractal dimension of our spacetime diagram we use the transition matrix of the matrix substitution system, which contains the information about the images of all states. The line corresponding to F would contain a 1 in the rows of A, B, E, and F and zeros elsewhere. As every cell gives rise to four new cells the sum of all entries in each column of the matrix is 4. We thus deal with a sparse 256×256 matrix. The base 2 logarithm of the second largest eigenvalue of this matrix is the fractal dimension of the spacetime diagram (cf. for instance [Wil87]). Here this gives a fractal dimension of $\log_2 \frac{3+\sqrt{17}}{2} \simeq 1.8325$, as also found in [Mac04].

Let us note that up to this point our analysis for Θ is word for word valid for all CA in $\mathscr{M}_2(\mathbb{Z}_2[u,u^{-1}])$ of determinant 1 and trace $u^{-1}+1+u$. The additional information is only used for the actual colouring of the picture. In general all CA with the same minimal polynomial have the same substitution system, and in dimension 2 the minimal polynomial is entirely determined by the trace and the determinant. The fractal we get if we use the substitution system starting from $\begin{array}{|c|c|c|c|c|c|c|}\hline \cdots & 0 & H & G & F & E & 0 & \cdots\\ \hline \end{array}$ is shown in Figure 8.

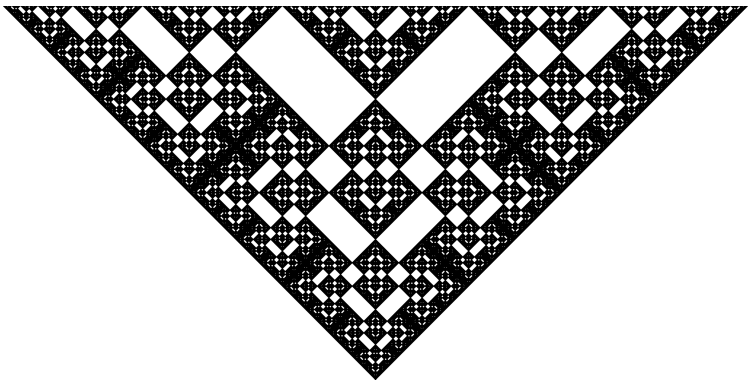


Figure 8: The general fractal time evolution of a linear CA with $k = m = 2$, determinant 1 and trace $u^{-1} + 1 + u$. Only the areas where the whole group of αs is 0 is marked white. Thus the image appears to have less white than the coloured picture. In the limit of infinite recursion this effect vanishes, thus the fractal that is generated is actually the same.

In the case of Θ the connection between the substitution system and the coloured picture is very simple; let us take $\xi = \binom{1}{0}$ as the initial state. Then the state of cell x after y iterations is $\Theta_x^y\xi = \sum_i \left(\alpha_{\Theta,0}(x-i,y)\Theta_i^0 + \alpha_{\Theta,1}(x-i,y)\Theta_i^1\right)\xi = \alpha_{\Theta,0}(x,y)\mathbb{1}\xi + \sum_i \alpha_{\Theta,1}(x-i,y)\Theta_i^1\xi$. Since $\Theta_0^1 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$, $\Theta_1^1 = \Theta_{-1}^1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$ and $\Theta_i^1 = 0$ when $i \notin \{-1;0;1\}$, we have $T_x^y\xi = \begin{pmatrix} \alpha_{\Theta,0}(x,y) \\ \alpha_{\Theta,1}(x,y) \end{pmatrix}$. This gives us a colour assignment for each state of the matrix substitution system, which corresponds to simply dropping all states that include neither B nor F.

We can now determine the average hue of the spacetime diagram making use of the eigenvector corresponding to the second largest eigenvalue of the transition matrix [Wil87]. Let us say $\binom{1}{0}$, $\binom{0}{1}$ and $\binom{1}{1}$ are respectively coded by the colours c_{10}, c_{01} and c_{11}; let c_q be the white colour. We determine which symbols of the alphabet belong to each of the colours by looking only at the part $\binom{\alpha_{\Theta,0}(x,y)}{\alpha_{\Theta,1}(x,y)}$. Then we just add up all the weights of symbols with the same color in the eigenvector. We get the following unnormalised coefficients: c_{10}: $2(4+\sqrt{17})$, c_{01}: $2(4+\sqrt{17})$, and c_{11}: $5+\sqrt{17}$.

In figure 1a, this colour code was used: $c_{10} =$, $c_{01} =$ and $c_{11} =$. We must therefore have the following average hue: .

Conclusion

We have shown that every cellular automaton inducing a morphism of abelian groups produces a selfsimilar spacetime diagram. We exhibited an algorithm taking as input the local transition rule and outputting a description of these patterns. We only studied the one-dimensional case in this article, but the analysis can be carried over to higher dimensions with not much

more ado. Instead of $\mathscr{M}_d\left(R[u,u^{-1}]\right)$, a n-dimensional linear cellular automaton would then be an element of $\mathscr{M}_d(R[u_1,u_1^{-1},u_2,u_2^{-1},\dots u_n,u_n^{-1}])$, matrix substitution systems would become $(n+1)$-dimensional array substitution systems, the system of indices in section 3 would be further complicated, and the spacetime diagrams would be harder to display. However, the generalisation does not present any theoretical difficulty.

We list here some open questions and possible future developments.

- Possibly, the m in Theorem 1 can always be taken to be 1. This is known to be true in the cyclic case, i.e. when $d = 1$, cf [Tak92].
- The algorithm presented in this article, producing a description of the spacetime diagram in the form of a matrix substition system, has a high complexity, due to the large size of its output. Is this a necessary evil, or can more efficient descriptions be found? Could for instance the more elegant triangle-based substitution scheme presented in section 2 be naturally generalised?
- To what extent can the algebraic structure be weakened? Instead of the alphabet being an abelian group, could we consider an abelian monoid? Is it possible to get rid of commutativity and/or associativity?

Acknowledgements

The authors would like to thank Jean-Paul Allouche, Bruno Durand, Cris Moore and Volkher Scholz for their useful feedback and bibliographical hints. They also gratefully acknowledge the support of the Deutsche Forschungsgemeinschaft (Forschergruppe 635), the EU (projects CORNER and QICS), the Erwin Schrödinger Institute and the Rosa Luxemburg Foundation.

References

[AHP+97] Jean-Paul Allouche, Fritz von Haeseler, Heinz-Otto Peitgen, A. Petersen, and Guentcho Skordev. Automaticity of double sequences generated by one-dimensional linear cellular automata. *Theoretical Computer Science*, 188(1–2):195–209, November 1997.

[AHPS96] Jean-Paul Allouche, Fritz von Haeseler, Heinz-Otto Peitgen, and Guentcho Skordev. Linear cellular automata, finite automata and Pascal's triangle. *Discrete Applied Mathematics*, 66(1):1–22, April 1996.

[Güt10] Johannes Gütschow. Entanglement generation of Clifford quantum cellular automata. *Applied Physics B*, 98(4):623–633, March 2010.

[GUWZ10] Johannes Gütschow, Sonja Uphoff, Reinhard F. Werner, and Zoltán Zimborás. Time asymptotics and entanglement generation of Clifford quantum celluar automata. *Journal of Mathematical Physics*, 51(1), January 2010.

[HPS93] Fritz von Haeseler, Heinz-Otto Peitgen, and Guentcho Skordev. Cellular automata, matrix substitutions and fractals. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):345–362, September 1993.

[HPS01] Fritz von Haeseler, Heinz-Otto Peitgen, and Guentcho Skordev. Self-similar structure of rescaled evolution sets of cellular automata. *International Journal of Bifurcation and Chaos*, 11(4):913–941, 2001.

[Lan93] Serge Lang. *Algebra*. Addison-Wesley publishing company, third edition, 1993.

[Mac04] Alan J. Macfarlane. Linear reversible second-order cellular automata and their first-order matrix equivalents. *Journal of Physics A: Mathematical and General*, 37:10791–10814, 2004.

[Mac09] Alan J. Macfarlane. On the evolution of the cellular automaton of rule 150 from some simple initial states. *Journal of Mathematical Physics*, 50(6):062702, 2009.

[MGAP85] Benoît B. Mandelbrot, Yuval Gefen, Amnon Aharony, and Jacques Peyrire. Fractals, their transfer matrices and their eigen-dimensional sequences. *Journal of Physics A: Mathematical and General*, 18:335–354, 1985.

[Moo97] Cristopher Moore. Quasi-linear cellular automata. *Physica D*, 103:100–132, 1997.

[Moo98] Cristopher Moore. Non-abelian cellular automata. *Physica D*, 111:27–41, 1998.

[MOW84] Olivier Martin, Andrew M. Odlyzko, and Stephen Wolfram. Algebraic properties of cellular automata. *Communications in Mathematical Physics*, 93:219–258, March 1984.

[SVW08] Dirk M. Schlingemann, Holger Vogts, and Reinhard F. Werner. On the structure of Clifford quantum cellular automata. *Journal of Mathematical Physics*, 49, 2008.

[Tak90] Satoshi Takahashi. Cellular automata and multifractals: dimension spectra linear cellular automata. *Physica D*, 45(1-3):36–48, 1990.

[Tak92] Satoshi Takahashi. Self-similarity of linear cellular automata. *Journal of Computer and System Sciences*, 44(1):114–140, 1992.

[Wil87] Stephen J. Willson. Computing fractal dimensions for additive cellular automata. *Physica D*, 24:190–206, 1987.

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 75–94

The Size of One-Way Cellular Automata

Martin Kutrib[1] and Jonas Lefèvre[2] and Andreas Malcher[1]

[1] *Institut für Informatik, Universität Giessen*
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher}@informatik.uni-giessen.de
[2] *Ecole Normale Supérieure de Lyon*
46 Allée d'Italie, 69362 Lyon, France
jonas.lefevre@ens-lyon.fr

We investigate the descriptional complexity of basic operations on real-time one-way cellular automata with an unbounded as well well as a fixed number of cells. The size of the automata is measured by their number of states. Most of the bounds shown are tight in the order of magnitude, that is, the sizes resulting from the effective constructions given are optimal with respect to worst case complexity. Conversely, these bounds also show the maximal savings of size that can be achieved when a given minimal real-time OCA is decomposed into smaller ones with respect to a given operation. From this point of view the natural problem of whether a decomposition can algorithmically be solved is studied. It turns out that all decomposition problems considered are algorithmically unsolvable. Therefore, a very restricted cellular model is studied in the second part of the paper, namely, real-time one-way cellular automata with a fixed number of cells. These devices are known to capture the regular languages and, thus, all the problems being undecidable for general one-way cellular automata become decidable. It is shown that these decision problems are NLOGSPACE-complete and thus share the attractive computational complexity of deterministic finite automata. Furthermore, the state complexity of basic operations for these devices is studied and upper and lower bounds are given.

Keywords: cellular automata, state complexity, descriptional complexity, formal languages, decidability

1 Introduction

Cellular automata are a well-motivated and well-investigated model for massively parallel computations which have widely been investigated from a computational capacity point of view (see, for example, the surveys [9, 10]). Basically, one-way cellular automata are linear arrays of identical copies of deterministic finite automata, sometimes called cells, that work synchronously at discrete time steps. Each cell is connected to its immediate neighbors to the right. The input is initially written into the cells.

Though real-time one-way cellular automata are one of the weakest classes of cellular automata, the class of languages accepted by them contains rather complicated non-context-free and non-semilinear languages and almost all important decidability questions turned out to be undecidable [16] and not even semidecidable [12].

Opposed to the computational capacity and complexity the descriptional complexity concerns the size of a system. One typical question is, for example, how succinctly a real-time one-way cellular automaton can represent a language in comparison with other models. In [4, 6] more general introductions to and surveys of descriptional complexity are given. The descriptional complexity of real-time one-way cellular automata and the related model of real-time iterative arrays has been studied in [12, 14].

An important branch of descriptional complexity is the study of the complexity of operations. Here, one considers language operations such as union, intersection, or reversal, under which the language classes of the devices in question are closed. Of interest are optimal constructions with regard to the size of description. Thus, the goal is to find upper bounds that give the sufficient size necessary to represent the result of applying an operation, and lower bounds that give the sizes necessary in the worst case. Since, in general, the minimization or even reduction of the size for a given one-way cellular automaton is algorithmically unsolvable, there is no general method to prove the minimality of a given device. Moreover, the precise upper bounds on the size may depend on undecidable properties. So, tight bounds in the order of magnitude are to some extent best possible.

There are many ways to measure the size of a system. For deterministic finite automata the number of states is a reasonable and popular measure. Since basically the representation of a cellular automaton consists of the representation of their cells, the number of states is a reasonable size measure for cellular automata, too. For deterministic and nondeterministic finite automata the state complexity of many operations has been investigated. Recent surveys of results with regard to deterministic finite automata are [19, 20], where also operations on unary regular languages are discussed. In [1] an automata independent approach based on derivatives of languages is presented, that turned out to be a very useful technique for proving upper bounds for deterministic finite automata operations (cf. [2, 3]). A systematic study of language operations in connection with nondeterministic finite automata is [5]. The operation problem for two-way deterministic finite automata has been investigated recently in [8].

In this paper, we study the state complexity of real-time one-way cellular automata and we consider the Boolean operations union, intersection, and complementation as well as the operation of reversal. We obtain upper and lower bounds which are tight in order of magnitude. Interestingly, the state complexity of the Boolean operations is very similar to that of deterministic finite automata. This is not longer true for the operation of reversal. Here, deterministic finite automata have an exponential blow-up whereas the blow-up for real-time one-way cellular automata is at most quadratic. In contrast to the composition of two languages by using an operation, the somehow "inverse" problem of decomposing a given language into two languages with the help of an operation is studied. Clearly, the goal is to find a shorter representation of the given language by decomposition. It turns out that such decomposition problems are algorithmically unsolvable with regard to the Boolean operations and reversal.

These undecidability results together with the undecidability of almost all commonly investigated questions motivates the study of a restricted model, the real-time one-way cellular automata with a fixed number of cells, which have been introduced in [13]. The computational power of the restricted model is equivalent to the regular languages and, thus, all the problems undecidable for general one-way cellular automata become decidable. It is shown that these decision problems are NLOGSPACE-complete and thus share the attractive computational complexity of deterministic finite automata. Furthermore, the state complexity of basic operations for these devices is studied and upper and lower bounds are given.

2 Definitions

We denote the positive integers and zero $\{0, 1, 2, ...\}$ by $\mathbb{N}$. The empty word is denoted by λ, the reversal of a word w by w^R, and for the length of w we write $|w|$. For the number of occurrences of a subword x in w we use the notation $|w|_x$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions. In order to avoid technical overloading in writing, two languages L and L' are considered to be equal, if they differ at most by the empty word, that is, $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$. Throughout the article two devices are said to be *equivalent* if and only if they accept the same language.

A one-way cellular automaton is a linear array of identical deterministic finite state machines, sometimes called cells. Except for the rightmost cell each one is connected to its nearest neighbor to the right. We identify the cells by positive integers. The state transition depends on the current state of a cell itself and the current state of its neighbor, where the rightmost cell receives information associated with a boundary symbol on its free input line. The state changes take place simultaneously at discrete time steps. The input mode for cellular automata is called parallel. One can suppose that all cells fetch their input symbol during a pre-initial step.

Definition 1 *A* one-way cellular automaton (*OCA*) *is a system* $\langle S, F, A, \#, \delta \rangle$*, where* S *is the finite, nonempty set of* cell states*,* $F \subseteq S$ *is the set of* accepting states*,* $A \subseteq S$ *is the nonempty set of* input symbols*,* $\# \notin S$ *is the permanent* boundary symbol*, and* $\delta : S \times (S \cup \{\#\}) \to S$ *is the* local transition function.

A *configuration* of a one-way cellular automaton $\langle S, F, A, \#, \delta \rangle$ at time $t \geq 0$ is a description of its global state, which is formally a mapping $c_t : \{1, 2, \ldots, n\} \to S$, for $n \geq 1$. The operation starts at time 0 in a so-called *initial configuration*, which is defined by the given input $w = a_1 a_2 \cdots a_n \in A^+$. We set $c_0(i) = a_i$, for $1 \leq i \leq n$. Successor configurations are computed according to the global transition function Δ. Let c_t, $t \geq 0$, be a configuration with $n \geq 2$, then its successor c_{t+1} is defined as follows:

$$c_{t+1} = \Delta(c_t) \iff \begin{cases} c_{t+1}(i) = \delta(c_t(i), c_t(i+1)), i \in \{1, 2, \ldots, n-1\} \\ c_{t+1}(n) = \delta(c_t(n), \#) \end{cases}$$

For $n = 1$, the next state of the sole cell is $\delta(c_t(1), \#)$. Thus, Δ is induced by δ.

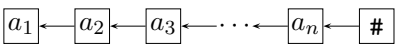


Fig. 1: A one-way cellular automaton.

An input w is accepted by an OCA $\mathcal{M}$ if at some time step during the course of its computation the leftmost cell enters an accepting state. The *language accepted by* $\mathcal{M}$ is denoted by $L(\mathcal{M})$. Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping. If all $w \in L(\mathcal{M})$ are accepted with at most $t(|w|)$ time steps, then $\mathcal{M}$ is said to be of time complexity t.

Observe that time complexities do not have to meet any further conditions. This general treatment is made possible by the way of acceptance. An input w is accepted if the leftmost cell enters an accepting state at some time $i \leq t(|w|)$. Subsequent states of the leftmost cell are not relevant. However, in the sequel we are particularly interested in fast OCAs operating in *real-time*, that is, obeying the time complexity $t(n) = n$.

So, any OCA is defined by the state set S, the set of input symbols A, the set of accepting states F, and the transition function. That means, for n states we have at most $2^n \cdot 2^n \cdot n^{n(n+1)}$ different OCAs, where, in addition, some of them are isomorphic. Since there are infinitely many languages acceptable by real-time OCAs, trivially, the number of states has to be unbounded.

3 State Complexity of Basic Operations

We consider the state complexities of the Boolean operations and reversal under which the class of languages accepted by real-time one-way cellular automata is closed [16]. First, we provide exemplarily an infinite language family over a binary alphabet that requires growing size when accepted by real-time OCAs. These languages and variants thereof are of tangible advantage for our purposes. As mentioned before, the problem is to prove a lower bound for the number of states necessary, since no general tools are available. Our lower bound misses the upper bound by one state only.

For all integers $k \geq 2$ let

$$L_k = \{\, 0^i a^{j \cdot k^i} \mid i, j \geq 1 \,\}.$$

Lemma 2 *Let $k \geq 2$ be an integer. Then $k+4$ states are sufficient for a real-time OCA to accept L_k.*

Proof: The language L_k is accepted by the real-time OCA $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$, where $A = \{a, 0\}$, $S = \{a, 0, 1, \ldots, k-1, <, +, -\}$, $F = \{+\}$, and

$$\begin{array}{ll}
\delta(a, \#) = - & \delta(p, a) = p+1,\ 0 \leq p \leq k-2 \\
\delta(a, -) = - & \delta(k-1, a) = < \\
\delta(<, -) = + & \delta(<, a) = 1 \\
\delta(<, +) = + & \delta(p, <) = p+1,\ 0 \leq p \leq k-2 \\
 & \delta(k-1, <) = < \\
 & \delta(<, q) = 0,\ 0 \leq q \leq k-1
\end{array}$$

Here and in the sequel we assume tacitly that the state is *not* changed whenever δ is not defined.

Basically, the idea of the construction is to set up a k-ary counter in the leftmost i cells, where states $0, 1, \ldots, k-1$ represent the digits and $<$ a carry-over to be processed by the left neighbor cell. The states $+$ and $-$ are used to implement a signal, which is initially started in the rightmost cell. It moves to the left, passes through the a-cells with a non-accepting state, and checks whether all cells of the counter passed through have been indicating a carry-over in the step before. Only in this case the accepting state is used. □

By almost the same reasoning the same upper bound for the complement $\overline{L_k}$ of L_k is shown.

Corollary 3 *Let $k \geq 2$ be an integer. Then $k+4$ states are sufficient for a real-time OCA to accept $\overline{L_k}$.*

Proof: We adapt the proof of Lemma 2 by defining $F = \{a, +\}$ and modifying δ such that $\delta(p, +) = +$, for all $p \in S$, $\delta(a, 0) = +$, $\delta(0, \#) = +$, $\delta(a, \#) = -$, $\delta(a, -) = -$, $\delta(<, -) = -$, and $\delta(p, -) = +$, for $0 \leq p \leq k-1$. □

Now we turn to the lower bounds.

Lemma 4 *Let $k \geq 2$ be an integer. Then at least $k+3$ states are necessary for a real-time OCA to accept L_k.*

Proof: Let $\mathcal{M}$ be a real-time OCA with state set S accepting L_k. We consider accepting computations on inputs of the form $0^i a^{j \cdot k^i}$ and, first, treat subcomputations as follows. The left part of sequences of adjacent a-cells runs through cycles according to $\delta(a,a) = a_1$, $\delta(a_1,a_1) = a_2$, $\delta(a_2,a_2) = a_3$, ... Denote the cycle length by c_a. Clearly, c_a is at most $|S|$. Therefore, for j large enough, the leftmost i cells initially carrying a 0 eventually also will run through cycles whose length is denoted by c_0. Finally, we have a possible signal from right to left initiated by $\delta(a,\#)$. Let $\delta(a,\#) = s_1$, $\delta(a_1,s_1) = s_2$, $\delta(a_2,s_2) = s_3$, ... Again, the signal eventually becomes cyclic with cycle length, say, c_s. Clearly, c_s is at most $|S|^2$.

Now we turn to states. Assume c_0 is at most $k^i - 1$. Since $0^i a^{k^i}$ is accepted, the input $0^i a^{k^i} a^{c_0 \cdot c_a \cdot c_s}$ must be accepted, too. But for i large enough, $(k^i - 1) \cdot c_a \cdot c_s = k^i \cdot c_a \cdot c_s - c_a \cdot c_s$ is not a multiple of k^i. Therefore, at least k states $z_1, z_2, \ldots, z_k$ are necessary to set up the cycle length c_0.

Furthermore, at least one state s_1 is necessary to realize the signal from right to left, where s_1 has to be different from a_1 and both are non-accepting states. Otherwise, there would be no signal and the whole computation could not accept in time. Clearly, neither $s_1, s_2, s_3, \ldots$ nor the states $a, a_1, a_2, a_3, \ldots$ and $z_1, z_2, \ldots, z_k$ can be accepting. So, in addition, one accepting state + is necessary.

If $a_1 \in \{z_1, z_2, \ldots, z_k\}$, at some time during the cycle the leftmost k cells are synchronously in state a_1 while further cells to their right are in state a_1 as well. So, some input belonging to L_k would be rejected. Similarly, if $s_1 \in \{z_1, z_2, \ldots, z_k\}$, then at some time the i-th cell from the left is in state s_1 and, thus, simulates the arrival of the signal, while the signal has not yet arrived. So, an input not belonging to L_k would be accepted.

Altogether, we have at least the $k+3$ states $\{z_1, z_2, \ldots, z_k, a_1, s_1, +\}$. □

As before, by almost the same reasoning the same lower bound for the complement $\overline{L_k}$ of L_k can be shown.

Corollary 5 *Let $k \geq 2$ be an integer. Then at least $k+3$ states are necessary for a real-time OCA to accept $\overline{L_k}$.*

3.1 Intersection and Union

Basically, the upper bounds for intersection and union are obtained by constructions based on the well-known two-track technique. That is, on two different tracks acceptors for both languages are simulated independent of each other. However, in general, an input is accepted when the leftmost cell enters an accepting state at some arbitrary time step. So, in general, the leftmost cell will enter accepting as well as non-accepting states during a computation. While this causes no problem for union, where a cell accepts when at least one of its registers is accepting, for the intersection, where a cell accepts when both of its registers are accepting, we have to provide further states. These are used to indicate that a register already has passed through an accepting state.

Theorem 6 *Let $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time OCA with r_1 non-accepting states, and $\mathcal{M}_2$ be an n-state real-time OCA with r_2 non-accepting states. Then $m \cdot n + r_1 \cdot n + m \cdot r_2 + r_1 \cdot r_2 \in O(m \cdot n)$ states are sufficient for a real-time OCA to accept $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$.*

Proof: We apply the two-track technique where each register remembers whether it has passed through an accepting state. First we modify $\mathcal{M}_i = \langle S_i, F_i, A, \#, \delta_i \rangle$, for $i \in \{1,2\}$ to $\hat{\mathcal{M}}_i = \langle \hat{S}_i, \hat{F}_i, A, \#, \hat{\delta}_i \rangle$, where $\hat{S}_i = S_i \cup R_i$ with $R_i = \{ s' \mid s \in S_i \setminus F_i \}$, $\hat{F}_i = F_i \cup R_i$, and

$$\hat{\delta}_i(s,t) = \begin{cases} \delta_i(s,t) & \text{if } s \in S_i \setminus F_i \\ \delta_i(s,t)' & \text{if } s \in F_i \text{ and } \delta_i(s,t) \in S_i \setminus F_i \\ \delta_i(s,t) & \text{if } s \in F_i \text{ and } \delta_i(s,t) \in F_i \end{cases},$$

$$\hat{\delta}_i(s',t) = \begin{cases} \delta_i(s,t)' & \text{if } s' \in R_i \text{ and } \delta_i(s,t) \in S_i \setminus F_i \\ \delta_i(s,t) & \text{if } s' \in R_i \text{ and } \delta_i(s,t) \in F_i \end{cases}.$$

Clearly, $\mathcal{M}_i$ and $\hat{\mathcal{M}}_i$ are equivalent. Now, $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$ accepts $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$, where $S = \hat{S}_1 \times \hat{S}_2$, $F = \hat{F}_1 \times \hat{F}_2$, and $\delta((s_1,s_2),(t_1,t_2)) = (\hat{\delta}_1(s_1,t_1), \hat{\delta}_2(s_2,t_2))$. □

For the union the construction is slightly simpler. In this case it is not necessary to remember whether a register has passed through an accepting state. Therefore, the next upper bound follows immediately.

Theorem 7 *Let $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time OCA and $\mathcal{M}_2$ be an n-state real-time OCA. Then $m \cdot n \in O(m \cdot n)$ states are sufficient for a real-time OCA to accept $L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$.*

Now we can utilize the languages L_k for showing lower bounds which are tight in the order of magnitude.

Theorem 8 *Let $m, n \geq 6$ be integers such that $m-4$ and $n-4$ are relatively prime. Then at least $(m-4)(n-4)+3 \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time OCA to accept the intersection of an m-state real-time OCA and an n-state real-time OCA language.*

Proof: Let $k = m-4$ and $\ell = n-4$ be relatively prime. The witness languages for the assertion are L_k accepted by an m-state real-time OCA and L_ℓ accepted by an n-state real-time OCA. The intersection $L_k \cap L_\ell$ is $L_{k \cdot \ell} = \{ 0^i a^{j \cdot k^i \cdot \ell^i} \mid i,j \geq 1 \}$. By Lemma 4, any real-time OCA accepting $L_{k \cdot \ell}$ has at least $k \cdot \ell + 3 = (m-4)(n-4)+3 \in \Omega(m \cdot n)$ states. □

Theorem 9 *Let $m, n \geq 6$ be integers such that $m-4$ and $n-4$ are relatively prime. Then at least $(m-4)(n-4)+3 \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time OCA to accept the union of an m-state real-time OCA and an n-state real-time OCA language.*

Proof: Let $k = m-4$ and $\ell = n-4$ be relatively prime. Now the witness languages for the assertion are $\overline{L_k}$ accepted by an m-state real-time OCA and $\overline{L_\ell}$ accepted by an n-state real-time OCA. Their union is $\overline{L_{k \cdot \ell}}$, for which at least $k \cdot \ell + 3 \in \Omega(m \cdot n)$ states are necessary by Corollary 5. □

3.2 Complementation

The precise upper bounds on the state complexity of some operations depend on the states that can appear on the diagonal of the space time diagram, that is, the states $c_i(n-i+1)$, $1 \leq i \leq n$. Given an OCA we consider the set of states D that can appear on the diagonal in some possible computation, and denote their number by d. For convenience, we simply write *states that can appear on the diagonal*.

For deterministic devices the closure under complementation is often shown by interchanging accepting and non-accepting states. The reason why this does not work in general for OCAs is once more that the leftmost cell may enter accepting as well as non-accepting states during a computation.

Theorem 10 *Let $n \geq 1$ be an integer and $\mathcal{M}$ be an n-state real-time OCA with r non-accepting states, d states that can appear on the diagonal and also at other positions, from which g are non-accepting. Then $n + r + d + g \in O(n)$ states are sufficient for a real-time OCA to accept $\overline{L(\mathcal{M})}$.*

Proof: We sketch the construction of a real-time OCA $\mathcal{M}'$ accepting $\overline{L(\mathcal{M})}$. Basically, $\mathcal{M}'$ simulates $\mathcal{M}$, but since the cells of $\mathcal{M}$ may enter accepting as well as non-accepting states during a computation, none of the states of $\mathcal{M}$ can be accepting in $\mathcal{M}'$. Instead, copies of the r non-accepting states are used in order to remember whether a cell has passed through an accepting state before. In order to accept the complement of $L(\mathcal{M})$ all of these new states are also non-accepting. Finally, it suffices to send a signal from right to left along the diagonal that causes every cell passed through that has not entered an accepting state before to accept. To this end, the states that appear on the diagonal have to be identified as signal. This is trivial for the states of $\mathcal{M}$ which appear only at the diagonal. For those d states that can appear on the diagonal and also at other positions (of $\mathcal{M}$), copies are used for this purpose. Furthermore, on the diagonal of $\mathcal{M}'$ there may appear g new non-accepting states indicating that the cell has entered an accepting state before. For these now new copies have to be used to accept. □

Theorem 11 *Let $n \geq 5$ be an integer. Then at least $2n - 3 \in \Omega(n)$ states are necessary in the worst case for a real-time OCA to accept the complement of an n-state real-time OCA language.*

Proof: For $k \geq 2$ the assertion is witnessed by the language

$$L_{c,k} = \{\, 0^i a^j \mid i \geq 1, j \geq k^i \,\}.$$

First, we construct a $(k+3)$-state real-time OCA $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$, which accepts it: $A = \{a, 0\}$, $S = \{0, 1, \ldots, k-1, a, \mathsf{<}, \mathsf{-}\}$, $F = \{\mathsf{<}\}$, and

$$\begin{array}{ll}
\delta(p, a) = p + 1,\ 0 \leq p \leq k-2 & \delta(k-1, \mathsf{<}) = \mathsf{<} \\
\delta(k-1, a) = \mathsf{<} & \delta(\mathsf{<}, q) = 0,\ 0 \leq q \leq k-1 \\
\delta(\mathsf{<}, a) = 1 & \delta(a, \#) = \mathsf{-} \\
\delta(p, \mathsf{<}) = p + 1,\ 0 \leq p \leq k-2 & \delta(a, \mathsf{-}) = \mathsf{-}
\end{array}$$

So, the real-time OCA $\mathcal{M}$ has $n = k + 3$ states, $r = k + 2$ non-accepting states, $d = k + 1$ states that can appear on the diagonal and at other positions, from which $g = k$ are non-accepting.

In order to show the lower bound on the number of states necessary to accept the complement of $L_{c,k}$, we argue as follows. At least k states are necessary to set up a cycle of length k^i in the i leftmost cells (cf. proof of Lemma 4). Clearly, these states are all non-accepting. Going into further details, at time step k cell i has to switch to a different set of at least k states. This is caused by the fact that the cycle of all the leftmost cells has to continue and, in addition, cell i can only change to an accepting state until time step k. In general, cell $1 \leq j \leq i$ has to switch to the different set of k states at time step $k^{i-j+1} + i - j$. Again, these new states are all non-accepting.

Furthermore, one additional state different from a is necessary to send a signal from right to left such that some cell initially carrying a 0 can change to an accepting state at all. Finally, an accepting state itself is necessary. In total, at least $2k + 3 = 2n - 3 \in \Omega(n)$ states are necessary. □

3.3 Reversal

Now we turn to the non-Boolean operation reversal.

Theorem 12 *Let $n \geq 1$ be an integer and $\mathcal{M}$ be an n-state real-time OCA with set of input symbols A and set D of states that can appear on the diagonal. Then $n \cdot |D| + |A| + 3 \in O(n^2)$ states are sufficient for a real-time OCA to accept $L(\mathcal{M})^R$.*

Proof: Let $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$ be real-time OCA with set D of states that can appear on the diagonal. In order to obtain a real-time OCA $\mathcal{M}'$ for the language $L(\mathcal{M})^R$, basically, the arguments of the local transition function are interchanged. In addition, we have to pay special attention to the boundary state. Moreover, $\mathcal{M}'$ cannot simulate the last step of $\mathcal{M}$ (see Figure 2). So, the construction has to be extended slightly. Each cell has an extra register that is used to simulate transitions of $\mathcal{M}$ under the assumption that the cell is the leftmost one. The transitions of the real leftmost cell now correspond to the missing transitions of the previous simulation. However, the computation of the leftmost cell of $\mathcal{M}$ is simulated on the diagonal of $\mathcal{M}'$ together with the additional register. So, if an accepting state appears on the diagonal, it has to be sent to the left. On the other hand, if an accepting state appears in the additional register, it has to cause the cell to accept but must not be sent to the left. So, we conclude the construction of $\mathcal{M}'$ by providing a signal from right to left which collects the results, where state + is the accepting state to be sent to the left, $\oplus$ is the accepting state not to be sent to the left, and - is the non-accepting state of the signal. Formally, $\mathcal{M}' = \langle S', F', A, \#, \delta' \rangle$ is constructed as follows.

$S' = (D \times S) \cup A \cup \{+, \oplus, -\}$, $F' = \{+, \oplus\}$,

for all $s_1, s_2 \in A$,

$$\delta'(s_1, s_2) = (\delta(s_1, \#), \delta(s_2, s_1)) \text{ and } \delta'(s_1, \#) = \begin{cases} + & \text{if } s_1 \in F \\ - & \text{if } s_1 \notin F \end{cases},$$

for all $d_1, d_2 \in D$, $s_1, s_2 \in S$,

$\delta'((d_1, s_1), (d_2, s_2)) = (\delta(s_1, d_1), \delta(s_2, s_1))$,

$\delta'((d_1, s_1), +) = +$,

$$\delta'((d_1, s_1), -) = \delta'((d_1, s_1), \oplus) = \begin{cases} + & \text{if } s_1 \in F \\ \oplus & \text{if } s_1 \notin F \text{ and } \delta(s_1, d_1) \in F \\ - & \text{otherwise} \end{cases}.$$

□

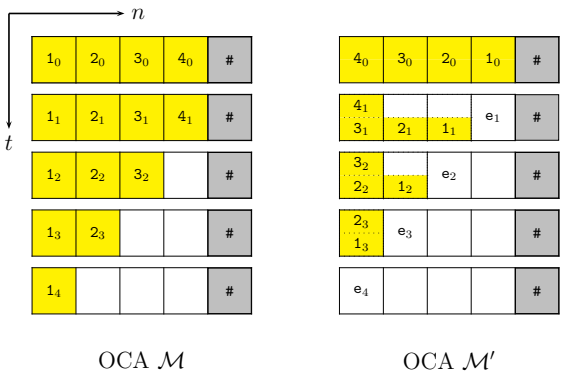


Fig. 2: Construction showing the simulation of a real-time OCA $\mathcal{M}$ by a real-time OCA $\mathcal{M}'$ on reversed input. The states e_1, e_2, e_3 are from $\{+, \oplus, -\}$. State e_4 depends on 1_3 and $1_4 = \delta(1_3, 2_3)$.

Theorem 13 *Let $k \geq 2$ and n be an integer of the form $12k+7$. Then at least $\Omega(n^2)$ states are necessary in the worst case for a real-time OCA to accept the reversal of an n-state real-time OCA language.*

Proof: For $k \geq 2$, the witness language for the assertion is

$$L_{R,k} = \{\, w0 \mid w \in \{a,b\}^*, |w| \geq k^2, |w|_a \equiv 0 \bmod k, |w|_b \equiv 0 \bmod k \,\},$$

which is accepted by a $(12k+7)$-state real-time OCA.

The formal construction is given below. We start with the idea of the construction. All cells initially carrying an a or b, behave as follows. In a first register they shift their input successively to the left. In a second register, they remember their original input. In a third register they count modulo k the number of input symbols shifted through that correspond to their own input symbol, that is, an a-cell counts all incoming symbols a and its own a, a b-cell counts all incoming symbols b and its own b. This behavior is realized by the first group of transition rules below.

In addition, initially a k-ary counter with two digits is set up at the right end. The first digit is initialized by the transitions $\delta(a,0)$ or $\delta(b,0)$ while the second digit is initialized by the transition $\delta(0,\#)$. The counter moves to the left. In addition to counting, both digits have two further registers. One register of the first digit indicates by + or - whether the last a-cell passed through has counted a number of a-symbols that is congruent 0 modulo k. The other register does the same for b-cells. This behavior is realized by the second group of transition rules below.

In order to distinguish between the first and the second digit of the moving counter, the second digit is primed. On every step to the left, the cell carrying the second digit simply takes the contents of the indicator registers of the first digit into their own indicator registers, and counts until a carry-over appears. Subsequently, the digit changes to an indicator + in its counting register which says that the counter has passed through at least k^2 cells. So, the cell carrying the second digit is in an accepting state if and only if the indicator + is in all of its registers. The behavior of the second digit is realized by the third group of transition rules below.

More precisely, the language $L_{R,k}$ is accepted by the real-time OCA $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$, where $S = \{a,b,0\} \cup \{a,b\} \times \{a,b\} \times \{0,1,\ldots,k-1\} \cup \{+,-\} \times \{+,-\} \times \{0,1,\ldots,k-1,0',1',\ldots,(k-1)',+\}$,

$A = \{a, b, 0\}$, $F = \{(+, +, +)\}$, and, for $x, y, s, t \in \{a, b\}$, $p, q \in \{0, 1, \ldots, k-1\}$,

$$\begin{aligned}
&\delta(a,a) = (a, a, 2 \bmod k) && \delta((x,s,p),(y,t,q)) = (y, s, (p+1) \bmod k), \text{ if } s = y\\
&\delta(a,b) = (b, a, 1) && \delta((x,s,p),(y,t,q)) = (y, s, p), \text{ if } s \neq y\\
&\delta(b,b) = (b, b, 2 \bmod k)\\
&\delta(b,a) = (a, b, 1)
\end{aligned}$$

and for $x, s \in \{a, b\}$, $u, v \in \{+, -\}$, $p, q \in \{0, 1, \ldots, k-1\}$,

$$\begin{aligned}
&\delta(a,0) = (-, +, 1)\\
&\delta(b,0) = (+, -, 1)\\
&\delta((x,s,p),(u,v,q)) = \begin{cases} (+, v, (q+1) \bmod k) & \text{if } s = a \text{ and } p = 0\\ (-, v, (q+1) \bmod k) & \text{if } s = a \text{ and } p \neq 0\\ (u, +, (q+1) \bmod k) & \text{if } s = b \text{ and } p = 0\\ (u, -, (q+1) \bmod k) & \text{if } s = b \text{ and } p \neq 0 \end{cases}
\end{aligned}$$

and for $u, v, w, z \in \{+, -\}$,

$$\begin{aligned}
&\delta(0, \#) = (+, +, 0')\\
&\delta((u,v,p),(w,z,q')) = (u, v, q'),\ 1 \leq p \leq k-1, 0 \leq q \leq k-1\\
&\delta((u,v,0),(w,z,q')) = (u, v, (q+1)'),\ 0 \leq q \leq k-2\\
&\delta((u,v,0),(w,z,(k-1)')) = (u, v, +)\\
&\delta((u,v,p),(w,z,+)) = (u, v, +),\ 0 \leq p \leq k-1
\end{aligned}$$

Without further proof we state that any real-time OCA accepting the reversal $L^R_{R,k}$ needs at least $\Omega(k^2) = \Omega(n^2)$ states. □

Since the upper bounds on the state complexity of complementation and reversal depend on the states that can appear on the diagonal of the space time diagram, it is natural to ask whether the constructions are effective. That is, to ask whether it is decidable which states appear on the diagonal. More general, the decidability of reachability problems such as whether there is an input and a time step at which a given configuration is reached by a given real-time OCA, or at which time a certain cell enters a given state, are of particular interest. We will show that the first question is decidable whereas the latter is undecidable.

Lemma 14 *Let $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$ be a real-time OCA, $n \geq 1$ be an integer, and $c : \{1, 2, \ldots, n\} \to S$ be a configuration. Then it is decidable whether there is an input $w \in A^n$ such that $\mathcal{M}$ reaches c on input w.*

Proof: We consider a brute-force algorithm which generates successively all inputs of length n, simulates the computation of $\mathcal{M}$ on these inputs until it becomes cyclically at latest at time step $|S|^n$, and finally checks whether the given configuration c occurred. As soon as such an input has been identified, the algorithm stops and returns *yes*. Otherwise, the algorithm stops after having negatively checked all possibilities and returns *no*. □

Lemma 15 *Let $\mathcal{M} = \langle S, F, A, \#, \delta \rangle$ be a real-time OCA, s be a state from S, and $i \geq 1$ be a cell. Then it is undecidable whether there is an input $w \in A^*$ such that on input w cell i enters state s at some time $t \geq 0$.*

Proof: Assume that the question is decidable. Then we can check for every accepting state $s \in F$ whether there is an input such that leftmost cell $i = 1$ enters s at some time $t \geq 0$. If this is not true for all $s \in F$, then $L(\mathcal{M})$ is empty and $L(\mathcal{M})$ is not empty otherwise. But in [12, 16] it has been shown that it is undecidable whether or not a given real-time OCA accepts the empty language, a contradiction. □

In particular, the last lemma reveals that it is not decidable which states appear on the diagonal. So, the constructions relying on these states are not effective. However, the effectiveness can be obtained by using the whole state set instead. On the other hand, we have to pay with unnecessary additional states for the effectiveness. Thus, to some extent tight bounds in the order of magnitude are best possible.

4 Unsolvability of Decompositions

So far, we have derived tight bounds in the order of magnitude for the number of states we have to pay when applying operations on real-time OCAs. Conversely, these bounds also show the maximal number of states that can be saved when a given minimal real-time OCA is decomposed into smaller ones. For example, given a minimal n-state real-time OCA that is equivalently to be represented by the union of two smaller real-time OCAs, we know that the product of the sizes of the smaller devices is at least n. Therefore, at least $2\sqrt{n}$ states are necessary for the decomposition into two smaller devices. From this descriptional complexity point of view, natural problems concern the question of whether such decompositions can algorithmically be solved. Given a k-ary operation under which the family of real-time OCA languages is closed, does there exist an algorithm that decomposes any given real-time OCA into k smaller ones if such a decomposition exists? We refer to such problems as *operation decomposition problems*. It turns out that such algorithms cannot exist for the operations in question. The proofs are reductions of undecidability problems for real-time OCAs. In [12, 16] it has been shown that it is neither decidable whether a given real-time OCA accepts no input (emptiness) nor whether it accepts all inputs (universality).

Theorem 16 ([12, 16]) *The emptiness and universality problems for real-time OCAs are undecidable.*

Theorem 17 *The union decomposition problem for real-time OCAs is algorithmically unsolvable.*

Proof: In contrast to the assertion, we assume there is an algorithm that solves the union decomposition. We obtain a contradiction by showing that in this case the emptiness for real-time OCAs is decidable. Clearly, any OCA has at least as many states as input symbols. Moreover, there is an OCA accepting the empty language which has exactly as many states, where none of them is accepting.

In order to decide whether a given real-time OCA accepts no input, we proceed as follows. First we inspect the set of accepting states. If it is empty, the answer is *yes*. If it contains at least one input symbol, the answer is *no*. Otherwise we apply the union decomposition algorithm. If as a result the algorithm reports that there is no decomposition, the answer is *no*. If the algorithm results in two smaller OCAs, we recursively apply the decision process to these devices. Now the answer is *yes* if and only if both smaller OCAs accept the empty language.

Why does this procedure give the correct answer? This is evident for the cases where the set of accepting states is empty or contains at least one input symbol. If otherwise the union decomposition algorithm is applied, we know that there is at least one accepting non-input state. So, if the OCA accepts the empty language, there is always a possible decomposition into two smaller OCAs having only input states which are all non-accepting. □

The same result for the intersection decomposition problem follows dual to the proof of the union decomposition problem. Now, a reduction of the undecidability of universality is used. Note that there is an OCA accepting all inputs which has exactly as many states as input symbols all of which are accepting.

Theorem 18 *The intersection decomposition problem for real-time OCAs is algorithmically unsolvable.*

The next results concern the unary language operations complementation and reversal.

Theorem 19 *The complementation decomposition problem for real-time OCAs is algorithmically unsolvable.*

Proof: Assume in contrast to the assertion that there is an algorithm that solves the complementation decomposition. Given a real-time OCA $\mathcal{M}$, we apply the algorithm successively until it reports that there is no further decomposition, whereby the number of applications is counted. Then we inspect the result and determine whether it has as many states as input symbols and, if so, whether these are all accepting or all non-accepting. So, we can decide for the result whether it accepts the empty language or all inputs or another language. The result is equivalent to $\mathcal{M}$ if the number of applications is even. Therefore, in this case we know whether $\mathcal{M}$ accepts the empty language or all inputs or another language. If, on the other hand, the number of applications is odd, we know whether the complement of $\mathcal{M}$ accepts the empty language or all inputs or another language. So, we can decide emptiness and universality of $\mathcal{M}$, a contradiction. □

Theorem 20 *The reversal decomposition problem for real-time OCAs is algorithmically unsolvable.*

Proof: As in the proof of Theorem 19, given a real-time OCA $\mathcal{M}$ we apply the algorithm successively until it reports that there is no further decomposition. Then we inspect the result and decide whether it accepts the empty language or all inputs or another language. Since the reversal of the empty language is the empty language and the same for the language of all words, we can decide emptiness and universality of $\mathcal{M}$, a contradiction. □

5 Real-time OCAs With a Fixed Number of Cells

Since for real-time OCAs almost all classical decidability questions are undecidable [16] and not even semidecidable [12], real-time OCAs are on the one hand a powerful parallel model, but on the other hand very unwieldy from a practical perspective. It would be interesting to know which resources of real-time OCAs have to be restricted in order to obtain decidable questions. In [11] real-time OCAs with sparse communication have been investigated, but still a very small amount of information communicated in one time step suffices to yield undecidability of the above questions. Other resources to be bounded are

classically time and space. Obviously, real time is the minimum time needed for useful computations. Concerning space constraints, logarithmic or sublogarithmic space bounds have been investigated for Turing machines [17] and real-time iterative arrays, which differ from real-time OCAs by a sequential processing of the input. It has been shown for the latter model [15], that logarithmic space still leads to undecidability whereas sublogarithmic space reduces the computational capacity of the model to the regular languages. For real-time OCAs it is not clear yet how logarithmic or, in general, sublinear space bounds should by defined properly. One problem to overcome is that the restricted model should be not more powerful than the unrestricted model. Consider an intuitively defined real-time OCA on unary input which possesses a logarithmic number of cells depending on the length of the input. Then it would be possible to accept the non-regular language $\{ a^{2^n} \mid n \geq 1 \}$ by implementing a binary counter in the provided cells. Since the latter language cannot be accepted by any real-time OCA, we obtain a stronger model.

Thus, it might be useful to consider in a first step real-time OCAs with a fixed number of cells. This model has been introduced and investigated in [13] with regard to descriptional complexity aspects. Since the computational capacity of the model is equivalent to the regular languages, all above-mentioned decidability questions become decidable and it is particularly interesting to compare this parallel model for the regular languages with the classical model of deterministic finite automata (DFAs) from a descriptional complexity point of view. Here, we will complement the results shown in [13] by investigating the state complexity of the Boolean operations and reversal. Furthermore, the computational complexity of the decidable problems turns out not to be more complicated than that for deterministic finite automata.

A k cells one-way cellular automaton works similar to the unrestricted model, but the input is processed as follows. At the beginning all k cells are in the quiescent state. The rightmost cell is the cell receiving the input. At every time step one input symbol is processed. All other cells behave as usual. The input is accepted, if at some time step the leftmost cell enters an accepting state. Since the minimal time to read the input and to send all information from the rightmost cell to the leftmost cell is the length of the input plus k, we provide a special end-of-input symbol $\triangledown$ to the rightmost cell after reading the input.

Definition 21 *A* k cells one-way cellular automaton *(kC-OCA) is a tuple* $\mathcal{M} = \langle S, F, A, s_0, \triangledown, k, \delta_r, \delta \rangle$ *where* S *is the finite, nonempty set of* cell states*,* $F \subseteq S$ *is the set of* accepting states*,* A *is the nonempty set of* input symbols*,* $s_0 \in S$ *is the* quiescent state*,* $\triangledown \notin S \cup A$ *is the* end-of-input symbol*,* k *is the* number of cells*, and* $\delta_r : S \times (A \cup \{\triangledown\}) \to S$ *is the* local transition function for the rightmost cell*, satisfying* $\delta_r(s_0, \triangledown) = s_0$*, and* $\delta : S \times S \to S$ *is the* local transition function for the other cells*, satisfying* $\delta(s_0, s_0) = s_0$*.*

A configuration of a kC-OCA at some time step $t \geq 0$ is a pair (c_t, w_t), where $w_t \in A^*$ denotes the remaining input and c_t is a description of the k cell states, formally a mapping $c_t : \{1, 2, \ldots, k\} \to S$. For an input $w = a_1 a_2 \cdots a_n \in A^*$ the initial configuration at time 0 is defined by $c_0(i) = s_0$, $1 \leq i \leq k$ and $w_0 = w$. Successor configurations are computed according to the global transition function Δ. Let (c_t, w_t), $t \geq 0$, be a configuration, then its successor configuration is defined as follows:

$$(c_{t+1}, w_{t+1}) = \Delta(c_t, w_t) \iff \begin{cases} c_{t+1}(i) = \delta(c_t(i), c_t(i+1)), i \in \{1, 2, \ldots, k-1\} \\ c_{t+1}(k) = \delta_r(c_t(k), a) \end{cases}$$

where $a = \triangledown$ and $w_{t+1} = \lambda$, if $w_t = \lambda$, and $a = a_1$ and $w_{t+1} = a_2 a_3 \cdots a_n$, if $w_t = a_1 a_2 \cdots a_n$. Thus, Δ is induced by δ_r and δ.

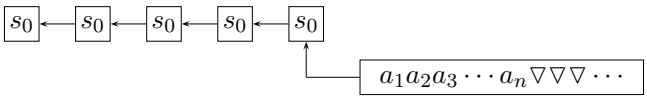


Fig. 3: Initial configuration of a 5 cells one-way cellular automaton (5C-OCA).

An input string w is accepted by a kC-OCA if at some time step during its computation the leftmost cell enters an accepting state. Real-time for kC-OCAs is defined as $|w|+k$ time steps.

Now, we investigate the state complexities of the Boolean operations and reversal for kC-OCAs and we start with two lemmas which will be useful to show lower bounds.

For all integers $k \geq 2$ and $\ell \geq 2$ let

$$L_{\ell,k} = \{\, a^i \mid i \equiv 0 \bmod \ell^k \,\}.$$

Lemma 22 *Let $k \geq 2$ and $\ell \geq 2$ be integers. Then $\ell+2$ states are sufficient for a real-time kC-OCA to accept $L_{\ell,k}$.*

Proof: To accept the language $L_{\ell,k}$ one has to set up an ℓ-ary counter in the k cells and to check, when the whole input has been read, whether the leftmost cell has generated a carry-over in the last but one time step. Thus, we need $\ell+1$ states to realize the ℓ-ary counter and one additional accepting state for the final check. Altogether, $\ell+2$ states are sufficient to accept $L_{\ell,k}$. □

Lemma 23 *Let $k \geq 2$ and $\ell \geq 2$ be integers. Then at least ℓ states are necessary for a real-time kC-OCA to accept $L_{\ell,k}$.*

Proof: Let $\mathcal{M}$ be a kC-OCA accepting $L_{\ell,k}$ with s states. We first show that $\mathcal{M}$ has to distinguish at least ℓ^k configurations. By way of contradiction, we assume that there are two different inputs a^n and a^m with $0 \leq n < m \leq \ell^k - 1$ leading to the same configuration c. From c we obtain on further input a^{ℓ^k-n} a configuration c'. Since $a^{n+\ell^k-n} = a^{\ell^k} \in L_{\ell,k}$, we have that $c'(1)$ is an accepting state. Then, $a^{m+\ell^k-n}$ belongs to $L_{\ell,k}$ as well. On the other hand, we can derive $0 < m-n < \ell^k$ which implies that $a^{m+\ell^k-n} \notin L_{\ell,k}$. This is a contradiction.

Hence, $\mathcal{M}$ must be able to represent at least ℓ^k different configurations and we obtain that $s^k \geq \ell^k$. Thus, $s \geq \ell$. □

5.1 Intersection and Union

The constructions for real-time kC-OCAs accepting the intersection or union of languages accepted by two given real-time kC-OCAs are very similar to the constructions for real-time OCAs given in Theorem 6 and Theorem 7. The constructions are again based on the two-track technique. Additionally, for intersection one has to keep track whether some register has already passed through an accepting state. Altogether, both constructions lead to the same bounds and we omit the details here.

Theorem 24 *Let $k \geq 2$ and $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time kC-OCA with r_1 non-accepting states, and $\mathcal{M}_2$ be an n-state real-time kC-OCA with r_2 non-accepting states. Then $m \cdot n + r_1 \cdot n + m \cdot r_2 + r_1 \cdot r_2 \in O(m \cdot n)$ states are sufficient for a real-time kC-OCA to accept $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$.*

Theorem 25 *Let $k \geq 2$ and $m, n \geq 1$ be integers, $\mathcal{M}_1$ be an m-state real-time kC-OCA and $\mathcal{M}_2$ be an n-state real-time kC-OCA. Then $m \cdot n \in O(m \cdot n)$ states are sufficient for a real-time kC-OCA to accept $L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$.*

Next, we will obtain that both upper bounds are tight in order of magnitude by showing the following lower bounds.

Theorem 26 *Let $k \geq 2$ be an integer and let $m, n \geq 4$ be integers such that m and n are relatively prime. Then at least $(m-2)(n-2) \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time kC-OCA to accept the intersection of an m-state real-time kC-OCA and an n-state real-time kC-OCA language.*

Proof: Let $m, n \geq 4$ be two integers which are relatively prime. We consider the languages $L_{m,k}$ and $L_{n,k}$ and obtain that $L_{m,k} \cap L_{n,k} = L_{mn,k}$. Due to Lemma 22 $L_{m,k}$ and $L_{n,k}$ can be accepted with $m+2$ and $n+2$ states, respectively. Owing to Lemma 23 we know that every real-time kC-OCA accepting the intersection $L_{mn,k}$ needs at least mn states. □

Theorem 27 *Let $k \geq 2$ and $m, n \geq 4$ be integers such that m and n are relatively prime. Then at least $(m-2)(n-2) \in \Omega(m \cdot n)$ states are necessary in the worst case for a real-time kC-OCA to accept the union of an m-state real-time kC-OCA and an n-state real-time kC-OCA language.*

Proof: Let $m, n \geq 4$ be two integers which are relatively prime. We consider the union of the languages $L_{m,k}$ and $L_{n,k}$. Due to Lemma 22 $L_{m,k}$ and $L_{n,k}$ can be accepted with $m+2$ and $n+2$ states, respectively.

Let $\mathcal{M}$ be a kC-OCA accepting $L_{n,k} \cup L_{m,k}$ with s states. It remains for us to show that $s \geq mn$. To this end, we prove that $\mathcal{M}$ has to distinguish at least $(mn)^k$ configurations. Then, we obtain that $s^k \geq (mn)^k$ which implies that $s \geq mn$.

By way of contradiction, we assume that there are two different inputs a^p and a^q with $0 \leq p < q \leq (mn)^k - 1$ leading to the same configuration c. Let $p' = p \bmod n^k$, $q' = q \bmod n^k$, $p'' = p \bmod m^k$, and $q'' = q \bmod m^k$. At first, we can show that $p' \neq q'$ or $p'' \neq q''$. Otherwise, we would have that $p' = q'$ and $p'' = q''$. Then, $p = t \cdot n^k + p'$ and $q = t' \cdot n^k + p'$ which implies that $q - p$ is a multiple of n^k. Analogously, we obtain that $q - p$ is a multiple of m^k. Thus, $q - p$ is a multiple of $(mn)^k$. This is a contradiction, since $0 < q - p < (mn)^k$.

From now on we assume without loss of generality that $p' \neq q'$ and $p' < q'$. Otherwise, we consider $p'' \neq q''$ or interchange the roles of p' and q' or p'' and q'', respectively. Let $0 < l \leq n^k$ be the unique integer such that $p'+l = n^k$. Then, $p+l \equiv 0 \bmod n^k$. Furthermore, we have that $n^k < q'+l < 2n^k$ which implies that $q+l \not\equiv 0 \bmod n^k$. Finally, we consider $q+l$ and distinguish two cases. If $q+l \not\equiv 0 \bmod m^k$, then we know that $a^{p+l} \in L_{n,k}$ and $a^{q+l} \notin L_{n,k} \cup L_{m,k}$. From configuration c we obtain on further input a^l a configuration c'. Since $a^{p+l} \in L_{n,k}$, we have that $c'(1)$ is an accepting state. Then, a^{q+l} belongs to $L_{n,k} \cup L_{m,k}$ as well which is a contradiction.

If $q + l \equiv 0 \bmod m^k$, then we know that $q'' + l = t''m^k$. Now, we consider $q'' + l + n^k$ and obtain that $q'' + l + n^k = t''m^k + n^k$ is not a multiple of m^k. Then, $q + l + n^k \not\equiv 0 \bmod m^k$. Moreover,

$q' + l + n^k$ is not a multiple of n^k, since $q' + l$ is not. Thus, $q + l + n^k \not\equiv 0 \bmod n^k$. Finally, $p' + l + n^k$ is a multiple of n^k, since $p' + l$ is. So, $p + l + n^k \equiv 0 \bmod n^k$ and we know that $a^{p+l+n^k} \in L_{n,k}$ and $a^{q+l+n^k} \notin L_{n,k} \cup L_{m,k}$. From configuration c we obtain on further input a^{l+n^k} a configuration c''. Since $a^{p+l+n^k} \in L_{n,k}$, we have that $c''(1)$ is an accepting state. Then, a^{q+l+n^k} belongs to $L_{n,k} \cup L_{m,k}$ as well which is a contradiction and concludes the proof. □

5.2 Complementation

The construction of a real-time kC-OCA accepting the complement of the language accepted by a given real-time kC-OCA is slightly different to the construction for real-time OCAs given in the proof of Theorem 10. However, the blow-up concerning the number of states is similar and we can show that the upper bound is tight in order of magnitude as well.

Theorem 28 *Let $k \geq 2$ and $n \geq 1$ be integers and $\mathcal{M}$ be an n-state real-time kC-OCA with r non-accepting states. Then $2(n + r) \in O(n)$ states are sufficient for a real-time kC-OCA to accept $\overline{L(\mathcal{M})}$.*

Proof: Let S and F denote the set of states and accepting states of $\mathcal{M}$, respectively. At first, we have to modify $\mathcal{M}$ such that $\mathcal{M}$ only accepts when the whole input has been processed. To this end, the rightmost cell emits a signal when it reads the end-of-input symbol for the first time. This signal moves to the left and remembers the state of the cell passed through, respectively. Finally, the signal will arrive at the leftmost cell exactly when the whole input has been processed and all information has been sent to the leftmost cell. At this time step we want to make the final decision whether to accept or to reject the input. So, the leftmost cell has to remember whether it has entered an accepting state at some time before. This can realized the same way as before by introducing a copy of the non-accepting states S' of the state set $S \setminus F$ of $\mathcal{M}$ and modifying the local transition function suitably. Then, the modified automaton $\hat{\mathcal{M}}$ accepts, if and only if the leftmost cell is in some state of $S' \cup F$ when the signal arrives. In order to accept the complement of $L(\mathcal{M}) = L(\hat{\mathcal{M}})$, it suffices to let the automaton accept, if and only if the leftmost cell is in some state of $S \setminus F$ when the signal arrives.

Disregarding the realization of the signal, the number of states needed is $n + r$. The implementation of the signal may at most double this number and we obtain $2(n + r)$ states as an upper bound. □

Theorem 29 *Let $k \geq 2$ and $n \geq 3$ be integers. Then at least $n - 1 \in \Omega(n)$ states are necessary in the worst case for a real-time kC-OCA to accept the complement of an n-state real-time kC-OCA language.*

Proof: We consider the witness languages

$$L'_{n,k} = \{\, a^i \mid i \geq n^k \,\}.$$

First, we construct an $(n+1)$-state real-time kC-OCA accepting $L'_{n,k}$. To this end, one has to set up an n-ary counter and to define the state denoting a carry-over as the only accepting state (see also Example 2.1 in [13]).

On the other hand, every kC-OCA accepting

$$\overline{L'_{n,k}} = \{\, a^i \mid i < n^k \,\}$$

needs at least n states, since at least n^k configurations have to be distinguished. □

5.3 Reversal

The construction of a real-time kC-OCA accepting the reversal of the language accepted by a given real-time kC-OCA is completely different to the construction for real-time OCAs given in the proof of Theorem 12 where a quadratic upper bound is shown. Here, we will obtain an exponential upper bound which is almost tight in order of magnitude.

Theorem 30 *Let $k \geq 2$ and $n \geq 1$ be integers and $\mathcal{M}$ be an n-state real-time kC-OCA. Then at most $2^{n^k - n^{k-1}+1} + 1 \in O(2^{n^k})$ states are sufficient for a real-time kC-OCA to accept $L(\mathcal{M})^R$.*

Proof: We present the intuitive construction. At first, we convert $\mathcal{M}$ to an equivalent DFA $\mathcal{N}$ having at most $n^k - n^{k-1} + 1$ states according to the construction given in [13]. Then, $\mathcal{N}$ is converted to a DFA $\mathcal{N}^R$ accepting the reversal of $L(\mathcal{N})$. By using the standard construction, $\mathcal{N}^R$ has at most $2^{n^k - n^{k-1}+1}$ states. Finally, $\mathcal{N}^R$ is converted to an equivalent kC-OCA $\mathcal{M}^R$. Due to the construction given in [13] we need one additional state which gives the upper bound claimed. □

The above construction arises the question whether it is in fact the best possible. In particular, the construction does not make use of the parallelism of kC-OCAs. The next lemma provides a lower bound which roughly says that the construction cannot be improved or parallelized essentially with regard to kC-OCAs. This shows that reversal is a very expensive operation for kC-OCAs whereas only a quadratic blow-up occurs for real-time OCAs.

Theorem 31 *Let $k \geq 2$ and $n \geq 3$ be integers such that $n \geq k$. Then at least $\Omega(2^{(n-1)^{k-1}})$ states are necessary in the worst case for a real-time kC-OCA to accept the reversal of an n-state real-time kC-OCA language.*

Proof: We consider the witness languages

$$L''_{n,k} = \{\, a^{n^k} \{a,b\}^i \mid i \geq 0 \,\}.$$

To accept the language $L''_{n,k}$, we can use the same construction as in the proof of Theorem 29. Thus, $L''_{n,k}$ can be accepted with $n+1$ states.

Now, let $\mathcal{M}$ be a kC-OCA accepting

$$L''^R_{n,k} = \{\, \{a,b\}^i a^{n^k} \mid i \geq 0 \,\}$$

with s states. We first show that $\mathcal{M}$ has to distinguish at least 2^{n^k} configurations. By way of contradiction, we assume that there are two different inputs $u, v \in \{a,b\}^{n^k}$ leading to the same configuration c. Since u and v are different, we obtain without loss of generality that $u = xaa^t$ and $v = yba^t$ with $0 \leq t \leq n^k - 1$. From configuration c we obtain on further input a^{n^k-t-1} a configuration c'. Since $ua^{n^k-t-1} \in L''^R_{n,k}$, we have that $c'(1)$ is an accepting state. Then, va^{n^k-t-1} belongs to $L''^R_{n,k}$ as well. This is a contradiction, since $va^{n^k-t-1} \notin L''^R_{n,k}$.

Since $\mathcal{M}$ must be able to represent at least 2^{n^k} different configurations, we obtain that $s^k \geq 2^{n^k}$. Thus, $s \geq 2^{\frac{n^k}{k}} \geq 2^{\frac{n^k}{n}} = 2^{n^{k-1}}$ since $n \geq k$. □

We may summarize the state complexity of the operations studied as follows. The state complexity of intersection and union for kC-OCAs of size m and n, respectively, is in $\Theta(mn)$. The state complexity of complementation for a kC-OCA of size n is in $\Theta(n)$. The upper bound of the state complexity of reversal for a kC-OCA of size n is in $O(2^{n^k})$ and the lower bound is in $\Omega(2^{(n-1)^{k-1}})$.

5.4 Computational Complexity

Finally, we discuss the computational complexity of typical decidability questions. For real-time OCAs these questions are known to be undecidable. Here, we show that the questions are decidable for kC-OCAs with $k \geq 2$ and, moreover, are NLOGSPACE-complete. Thus, the questions for kC-OCAs have the same computational complexity as for deterministic finite automata.

Theorem 32 *Let $k \geq 2$ be an integer. Then for kC-OCAs the problems of testing emptiness, universality, inclusion, and equivalence are NLOGSPACE-complete.*

Proof: First, we show that the problem of non-emptiness belongs to NLOGSPACE. Since NLOGSPACE is closed under complementation, emptiness belongs to NLOGSPACE as well. We describe a two-way nondeterministic Turing machine $\mathcal{M}$ which receives an encoding of some kC-OCA $\mathcal{A}$ on its read-only input tape and produces on its write-only output tape an answer *yes* or *no* while the space used on its working tape is bounded by $O(\log |cod(\mathcal{A})|)$. Then, the work space is bounded by $O(\log n)$ as well where n denotes the maximum of the number of states in $\mathcal{A}$ and the size of the input alphabet of $\mathcal{A}$, since both parameters are part of the encoding of $\mathcal{A}$ on the input tape of $\mathcal{M}$. It is shown in [13] that $\mathcal{A}$ can be converted to an equivalent DFA $\mathcal{A}'$ having at most $n^k - n^{k-1} + 1$ states. It has been shown in [7] by using the pumping lemma for regular languages that $L(\mathcal{A}')$ is not empty if and only if $L(\mathcal{A}')$ contains a word of length at most n^k. Thus, the idea for the Turing machine $\mathcal{M}$ is to guess a word of length at most n^k and to check whether it is accepted by $\mathcal{A}$. We implement on $\mathcal{M}$'s working tape a binary counter C which counts up to n^k. With the usual construction this needs at most $O(\log n^k) = O(k \log n) = O(\log n)$ tape cells. Additionally, we have to keep track of the current states of the k cells of $\mathcal{A}$. Clearly, the state of each cell can be represented by $O(\log n)$ tape cells. Altogether, a configuration of $\mathcal{A}$ can be represented by $O(\log n)$ tape cells. Now, $\mathcal{M}$ guesses one input symbol a, $\mathcal{M}$ increases the counter C, and updates all cells of $\mathcal{A}$ according to the transition function of $\mathcal{A}$ encoded on the input tape. This behavior is iterated until either the simulated leftmost cell of $\mathcal{A}$ enters an accepting state of $\mathcal{A}$ or the counter C has been counted up to n^k. In both cases $\mathcal{M}$ halts and outputs *yes* in the first case and outputs *no* in the latter. Altogether, $\mathcal{M}$ decides the non-emptiness of $\mathcal{A}$ and uses at most a logarithmic number of tape cells with regard to the length of the input.

For the problem of non-universality of a given kC-OCA $\mathcal{A}$ we test the non-emptiness of a kC-OCA $\mathcal{A}'$ accepting the complement of $L(\mathcal{A})$. The only difference to the above construction is that we have to simulate a computation in $\mathcal{A}'$ instead of $\mathcal{A}$. To this end, we consider the construction for the complement given in Theorem 28. Having programmed this modification of the transition functions of a kC-OCA in the finite control of the Turing machine $\mathcal{M}$ suitably, we can simulate a transition of $\mathcal{A}'$ when reading and translating a transition of $\mathcal{A}$ from the input tape. Additionally, we have to observe that the number of states of $\mathcal{A}'$ increases only by a linear factor of 4. Thus, it suffices for the counter C to count up to $(4n)^k$. Altogether, we obtain that non-universality is in NLOGSPACE. Due to the closure under complementation, universality is in NLOGSPACE as well.

The constructions for testing inclusion and equivalence are similar. For two kC-OCAs $\mathcal{A}_1$ and $\mathcal{A}_2$ we have that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ if and only if $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)}$ is empty. Due to the construction given in Theorem 24, we can reduce the question of inclusion to the question of testing the emptiness of a kC-OCA whose size is linearly bounded with regard to the size of $\mathcal{A}_1$ and $\mathcal{A}_2$. By similar observations as for non-universality, we obtain that the problem of inclusion is in NLOGSPACE. Finally, two kC-OCAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if and only if both $L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)}$ and $\overline{L(\mathcal{A}_1)} \cap L(\mathcal{A}_2)$ are empty. Thus, equivalence is in NLOGSPACE as well.

The hardness results follow directly from the hardness results for DFAs (see, e.g., the summary in [18]), since any DFA can be effectively converted to an equivalent kC-OCA [13] which simulates the given DFA in the rightmost cell and sends an additional accepting state to the leftmost cell when the end-of-input symbol is read and the input is accepted by the DFA. Obviously, this construction can be done in deterministic logarithmic space. □

References

[1] Brzozowski, J.: Quotient complexity of regular languages. In: Descriptional Complexity of Formal Systems (DCFS 2009), Otto-von-Guericke-Universität Magdeburg (2009) 25–42

[2] Brzozowski, J., Jirásková, G., Li, B.: Quotient complexity of ideal languages. In: Latin 2010: Theoretical Informatics. LNCS, Springer (2010) to appear

[3] Brzozowski, J., Jirásková, G., Zou, C.: Quotient complexity of closed languages. In: Computer Science Symposium in Russia (CSR 2010). LNCS, Springer (2010) to appear

[4] Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS **8** (2002) 193–234

[5] Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Int. J. Found. Comput. Sci. **14** (2003) 1087–1102

[6] Holzer, M., Kutrib, M.: Descriptional complexity – an introductory survey. In: Scientific Applications of Language Methods. Imperial College Press (2010) to appear

[7] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979)

[8] Jirásková, G., Okhotin, A.: On the state complexity of operations on two-way finite automata. In: Developments in Language Theory (DLT 2008). Volume 5257 of LNCS, Springer (2008) 443–454

[9] Kutrib, M.: Cellular automata – a computational point of view. In: New Developments in Formal Languages and Applications. Springer (2008) 183–227

[10] Kutrib, M.: Cellular automata and language theory. In: Encyclopedia of Complexity and System Science. Springer (2009) 800–823

[11] Kutrib, M., Malcher, A.: Cellular automata with sparse communication. In: Implementation and Application of Automata (CIAA 2009). Volume 5642 of LNCS, Springer (2009) 34–43

[12] Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom., Lang. Comb. **7** (2002) 549–560

[13] Malcher, A.: On one-way cellular automata with a fixed number of cells. Fund. Inform. **58** (2003) 355–368

[14] Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Trans. Inf. Syst. **E87-D** (2004) 721–725

[15] Malcher, A., Mereghetti, C., Palano, B.: Sublinearly space bounded iterative arrays. In: Automata and Formal Languages (AFL 2008), Hungarian Academy of Sciences (2008) 292–301

[16] Seidel, S.R.: Language recognition and the synchronization of cellular automata. Technical Report 79-02, Department of Computer Science, University of Iowa (1979)

[17] Szepietowski, A.: Turing Machines with Sublogarithmic Space. Volume 843 of LNCS, Springer (1994)

[18] Yu, S.: Regular languages. In: Handbook of Formal Languages. Volume 1. Springer (1997) 41–110

[19] Yu, S.: State complexity of regular languages. J. Autom., Lang. Comb. **6** (2001) 221–234

[20] Yu, S.: State complexity of finite and infinite regular languages. Bull. EATCS **76** (2002) 142–152

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 95–114

A weakly universal cellular automaton in the hyperbolic $3D$ space with three states

Maurice Margenstern[1]

[1] *Université Paul Verlaine − Metz, IUT de Metz,*
LITA EA 3097, UFR MIM,
Campus du Saulcy,
57045 METZ Cédex 1, FRANCE
e-mail: margens@univ-metz.fr
Web page: http;//www.lita.sciences.univ-metz.fr/~margens

In this paper, we significantly improve a previous result by the same author showing the existence of a weakly universal cellular automaton with five states living in the hyperbolic $3D$-space. Here, we get such a cellular automaton with three states only.

Keywords: universality, cellular automata, hyperbolic geometry, $3D$ space, tilings

1 Introduction

In this paper, we follow the track of previous papers by the same author, with various collaborators or alone, see [2, 7, 13, 14, 10, 9], which make use of the same basic model, the *railway model*, see [16, 5, 9]. In order to be within the space constraint for the paper, we just refer to the above mentioned paper both for what is the railway model and for what is hyperbolic geometrY. For the latter one, we just mention something new in Section 2. A more developped version of the paper can be found on *arXiv*, see [11].

In the previous papers, the number of states of a weakly universal cellular automaton was reduced from 24 states to 9 ones in the pentagrid and fixed at 6 for the heptagrid. In [10], I succeded to reduce this number to 4 in the heptagrid.

The reduction for 6 states to 4 states, using the same model, was obtained by replacing the implementation of the tracks of the railway model. In all previous papers, the track is implemented as a one-dimensional structure where each cell of the track has two other neighbours on the track exactly, considering that the cell also belongs to its neighbourhood. The locomotive follows the track by successively replacing two contiguous cells of the track: the cells occupied by the front and by the rear of the locomotive. The locomotive has its own colours and the track has another one which is also different from the blank, the colour of the quiescent state. In the mentioned paper, this traditional implementation is replaced by a new one. There, the track is no materialized but suggested only. It is delimited by *milestones* which may not define a continuous structure.

At this point, my attention was drawn by a referee of a submission to a journal explaining the 4-state result that it is easy to implement rule 110 in the heptagrid, using three states only. This is true, but this trick produces an automaton which is not really a planar automaton and does not improve our knowledge neither on rule 110 nor on cellular automata in the hyperbolic plane. This implementation with three states can also be easily adapted to the dodecagrid of the hyperbolic $3D$ space and suffers the same defect of bringing in no new idea.

In this paper, we follow the same idea of milestones as in [10]. Here too, the milestones are implemented in two versions. However, thanks to the third dimension, the same pattern can be used to change directions, either inside a plane of the hyperbolic $3D$ space or to switch from one plane to another one. This configuration is used to avoid crossings, replacing them by bridges, as this was already performed in [7]. Sections 3 and 4 thoroughly describe the implementation of the model in the hyperbolic $3D$ space. Section 5 explains how to check the rotation invariance of the rules. For the correctness of the rules themselves, we refer the reader to [11] where they are fully listed. In Section 5, we also give a short account on the computer program which we used to perform the simulation and to check the correctness of the rules.

This will conclude our proof of the following result:

Theorem 1 (Margenstern) – *There is a cellular automaton in the dodecagrid of the hyperbolic $3D$ space which is weakly universal and which has 3 states. Moreover, the cellular automaton is rotation invariant and its motion actually makes use of the three dimensions.*

By the latter expression, we mean that the automaton cannot be reduced to a lower dimension by a simple projection. We refer the reader to [9, 7] for a discussion on the notion of weak universality. The reader is also referred to [11] for figures and tables, not included here in order to comply to page constraints.

2 Navigation in the dodecagrid

Here, we use the ideas of [8, 9] to define navigation tools for the dodecagrid. Using Schlegel diagrams, see [7, 8, 9], we can also define a splitting of the hyperbolic $3D$ space into 8 corners around a point which is a common vertex. Next, we split the corner as indicated in Figure 1.

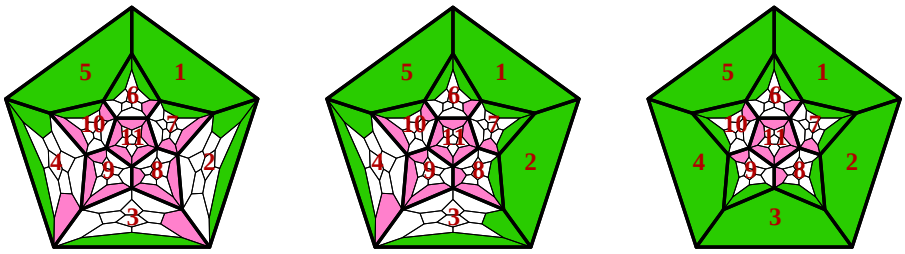


Figure 1 *Splitting a corner of the dodecagrid. On the left-hand side, the splitting of a corner. On the middle, the splitting of a half-octant. On the right-hand side, the splitting of a tunnel. Note that the faces are numbered according to the convention introduced in* [7, 9].

The idea of the representation is, as in the pentagrid, to fix rules which allow to get a bijection of a tree with the tiling restricted to the corner. If we allow the reflection of any dodecahedron of the tiling in its faces, we shall get many doubled replications as explained in [8].

We refer the reader to [8]. However, the splitting suggested by Figure 1 is a bit different from that indicated in [8]. The difference is that the splitting of Figure 1 is more symmetric and it involves three basic regions instead of four ones in the splitting of [8].

In the paper, we shall not directly use the tree. Taking into account that most of the circuitry will occur in a plane Π_0, we shall use projections onto Π_0. Now, we can chose Π_0 to be plane of a face of a fixed dodecahedron. In this way, the restriction of the dodecagrid to Π_0 is a copy of the pentagrid. And so, for the projections we have in mind, we can use the pentagrid.

In Π_0, each tile of the pentagrid is a face of exactly one tile of the dodecagrid over Π_0. We draw a Schlegel diagram of the corresponding dodecahedron within its face which lies on Π_0. We shall call this a **pseudo-projection onto** Π_0. Imagine that we have four tiles O, Y, G and B defined by their respective colour, orange, yellow, green and blue. Imagine that another tile W, a white one, sees Y through its face 1, the same face being numbered 5 in Y, the face 1 of Y being that which is shared with G. Then, we can see two other tiles on W, a red one and an orange one, on faces 10 and 6 respectively.

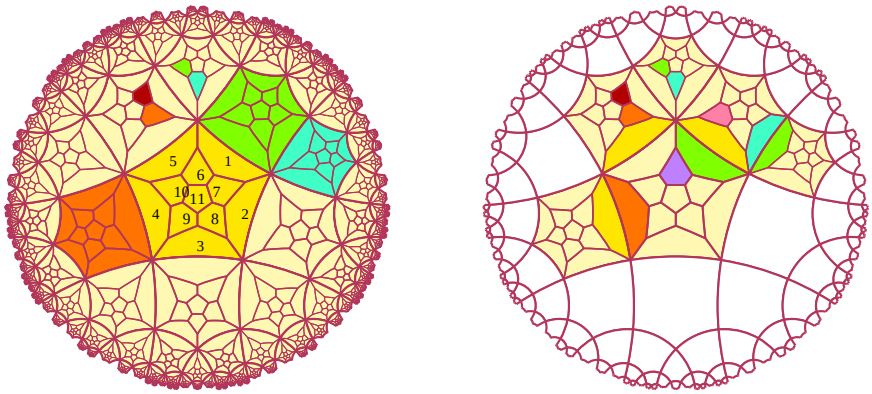


Figure 2 *Two different ways for representing a pseudo-projection on the pentagrid. On the left-hand side: the tiles have their colour. On the right-hand side: the colour of a tile is reflected by its neighbours only.*

On the left-hand side of Figure 2, the tiles keep their colour.

We can see that this raises a problem with the tiles which are put on the faces 6 and 10 of the tile W, in case W would be blue, for instance. For: what colour should be that of face 6? Will it be the colour of W or the colour of the other dodecahedron which shares it with W? Another problem is given, for instance by the faces 1 of Y and G, assuming that the face 1 of G is that which sees Y. In fact, as can easily be seen by the fact that these faces are both perpendicular to Π_0 and that they share a common edge lying in Π_0, these faces coincide. The fact that they have different colour might be misleading.

This can be avoided by fixing a convention. In order to keep as much information as possible in the pseudo-projection, we shall consider that a face of a tile does not show the colour of the tile but the colour of its neighbour sharing the same face. The right-hand side of Figure 2 shows the same configuration as the one of the left-hand side, but under the new convention. Also, to make the figure more readable, we do not draw the pseudo-projection of a tile who would be white with only white neighbours among those of its neighbours which do not touch Π_0. Now we can see that we can use the fact that two different faces coincide in the $3D$ space by indicating the colour of the other tile.

Later on, we shall adopt this second solution to represent our cellular automaton. Indeed, the cells of the cellular automaton are the dodecahedra of the dodecagrid and the colour of a tile is given by the state of the cellular automaton at the considered dodecahedron.

Before turning to the next section, we have an important remark.

We have already indicated that, in the pseudo-projection, faces which share an edge but which belong to different dodecahedra do coincide in the hyperbolic $3D$ space. The consequences are important with respect to the neighbours of a tile τ where, by neighbour, we mean a polyhedron which shares a face with τ. On the left-hand side part of Figure 2, we can see four small faces coloured with r, o, g and b on two white dodecahedra which we call W_1 and W_2, with W_1 being a neighbour of the central cell and W_2 a neighbour of the green neighbour of the central cell. We can view these coloured faces as dodecahedra obtained from the dodecahedron to which the face belongs by reflection in the very face. Call these dodecahedra by the colour of their defining faces. Considering the planes of the faces and their relations with Π_0, it is not difficult to see that the dodecahedra r and g are neighbours as well as the dodecahedra o and b. However, despite the fact that the corresponding faces share an edge, the dodecahedra r and o are not neighbours. However, as r, o and W_1 share a common edge, there is a fourth dodecahedron δ_1 sharing this edge which is not represented in the figure. Now, δ_1 plays an important role for both r and o as it is a neighbour for both of them. The same remark holds for the dodecahedra g and b for which there is a dodecahedron δ_2, a neighbour of both dodecahedra, sharing a common edge also with W_2. Moreover, it can be seen that δ_1 and δ_2 are also neighbours: their common face is in the plane of the common face of W_1 and W_2, which also contains the common face of r and g as well as the common face of o and b.

Both couples r with g and o with b can also be seen on the right-hand side part of Figure 2. There are also two other coloured small faces: a purple one on the central tile, call it p as well as the dodecahedron which it defines. There is also a pink one on the green dodecahedron which is a neighbour of the central one. Call the pink dodecahedron π. It is not difficult to see that the following pairs of dodecahedra are neighbours in the dodecagrid: b and π, π and p as well as p and o. Moreover, the four dodecahedra o, b, π and p share a common edge which belongs to the same line as the one which supports the edge shared by W_1, W_2, the central tile and G.

At last, remark that the plane of a face of a dodecahedron D defines two half-spaces: the half-space which does not contain D contains one neighbour of D exactly. The half-space which contains D contains all the other neighbours of D also. This can be seen as a consequence of the convexity of the dodecahedron.

Now, we can turn to the implementation of the railway model in the dodecagrid.

3 Implementation of the tracks

The implementation of the model is much more difficult in the hyperbolic $3D$-space than in the hyperbolic plane. Speaking about implementations in the hyperbolic plane, I often use the metaphor of a pilot flying with instruments only. This can be reinforced in the case of the hyperbolic $3D$-space by saying that this time we are in the situation of an astronaut who can do no other thing than fly with instruments only: sometimes, the astronaut may look at the earth. It is a fantastic image, however of no help for the navigation in cosmos. For the dodecagrid, we hope that the method explained in Section 2 shows that the situation is after all a bit better than in cosmos. The figures which we can obtain from the projections defined in Section 2 may help the reader to have a satisfactory view of the situation. We have to never forget that the views we can obtain are dramatically simplified images of what actually

happens. However, always bearing in mind that the images are always a local view, a good training based on rigorous principles may transform them into an efficient tool.

Remember that in most its parts, the track followed by the locomotive runs on a fixed plane of the hyperbolic $3D$ space. We shall see that we can assume that this plane is Π_0. Only occasionally, it switches to other planes, perpendicular to Π_0. In particular, this is the case for the implementation of crossings: as in [7], we take advantage of the third dimension in order to replace them by bridges. Also for the sensors which decorate the switches, we shall take benefit of the third dimension to differentiate the configurations of the various switches.

In this section, we deal with the tracks only, postponing the implementation of the switches to Section 4.

3.1 The pieces

Below, Figure 3 illustrates a copy of the most common element of the tracks, which we call the **straight element**. It consists of a single dodecahedron, the track itself, marked by four blue dodecahedra, the **milestones**, which are neighbours of this dodecahedron.

Note the numbering of the faces on the figure: it follows the convention mentioned in Section 2. In Figure 3, pictures (a) and (b), face 0 is not visible but it is visible in the other pictures. Similarly, face 5 and face 2 respectively, are not visible in pictures (c) with (d) and (e) with (f) respectively. Due to the role of the elements in the circuit, we shall say that face 1 is the **entry** of the element and that faces 3 and 4 are its **exits** in the case of pictures (a) and (b). In the case of pictures (c) with (d) and (e) with (f) respectively, the entries are face 4 with face 10 and face 3 with face 8 respectively. We shall say **exit 3**, **exit 4**, **exit 8** or **exit 10** if we need to make it more accurate. It is important to notice that exits and entries can be exchanged: we can have exit 1 and entry 3 but not exit 3 and entry 4. Such a change of direction is necessary, but it will be realized by another element. As the role of entry and exits can be exchanged, we shall use the word **exit** in general descriptions with the possible meaning of both an entry or an exit through the possibly indicated face.

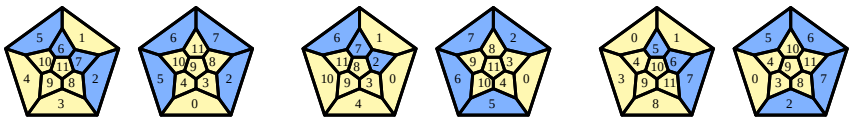


Figure 3 *An ordinary element of the track. Figure (a) is a view from above. Figure (b) is a view from the back of face 1. The locomotive enters the element via face 1 and exits via face 3 or face 4. It also may enter via face 3 or face 4 and then it exits through face 1.*

In Figures (c) and (d), the element is a bit turned around face 1 and the exits are now face (4) and (10). In Figures (e) and (f), the element is turned around face (1) too, but in the opposite direction, and the exits are now faces 3 and 8.

The motion in the opposite direction is always possible.

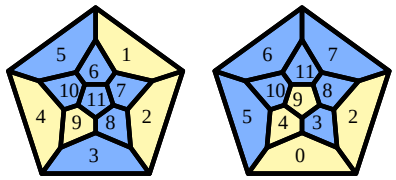
Remember the convention we introduced in Section 2. In most figures of the paper, if not otherwise mentioned, the colour of a cell can be deduced from the colours of the face of its neighbours. As an example, in the pictures of Figure 3, the milestones are blue and they are neighbours of the element.

As the name suggests, the milestones are usually fixed elements: they are not changed by the passage of the locomotive. This means that the milestones always remain blue, while the track is white as most cells of the space itself: the white state plays the role of the quiescent state: if a cell is white as well as all

its neighbours, then it remains white.

In Figure 3, the pictures represent various positions of the same elements which can be obtained from each other by a rotation a face of the dodecahedron or by a product of such rotations. We refer the reader to Subsection 5.1 where this problem is examined. In the figure, pictures (a) and (b) show a situation where Π_0 is the plane of face 0. In the pictures (c) and (d), it is that of face 5. In the pictures (e) and (f), it is that of face 2. The milestones can be viewed as the materialization of a catenary over the track itself, assumed to be put on the plane of the element.

Figure 4 illustrates another element of the track which we call a **corner**. This element allows the locomotive to perform a turn at a right angle. This possibility is very important and absolutely needed, as we shall see later.



(a) (b)

Figure 4 *The corner element of the track. Figure* (a) *is a view from above. Figure* (b) *is a view from the back of face* 1. *The locomotive enters the element via face* 1 *and exits via face* 2. *It also may enter via face* 2 *and then it exits through face* 1.

As we can see from Figure 4, the corner has more milestones around it then a straight element: 7 milestones instead of 4 ones. However, the face of a corner on Π_0 is white, while for a straight element the face on Π_0 is usually blue.

3.2 Vertical and horizontal segments

When finitely many straight elements are put one after each other, with the entry of one of them shared by the exit of the previous one, we say that these elements are set into a **vertical segment**, **vertical** for short, provided that the plane of these elements is the same and that there is a line of this plane which supports one side of each element which we call the **guideline**. Figure 5 illustrates the basic example of a vertical. The guideline supports a side of the faces 0 of the elements and the common plane is that of the faces 5.

In the representation of Figure 5, the dodecahedra are projected on the plane of face 5.

In the left-hand side picture of the figure, number the elements of the figure from 1 to 7. We can see that the element i is in contact with the element i+1 with $i \in \{1..6\}$. Consider elements 3 and 4, the latter one occupying the central pentagon of the picture. The exit 4 of element 4 and the entry 1 of element 3 appear as different faces of dodecahedra: each one is projected inside the face 5 of the dodecahedron. Now, by definition, the entry 1 of element 3 and the exit 4 of the element 4 coincide. Indeed: elements 3 and 4 have their faces 5 on a common plane. They also have their sides 0 on the guideline. The entry 1 of element 3 and the exit 4 of element 4 are perpendicular to the guideline and they share a common side: they are the same face.

As we stressed in Section 2, this situation is important and we shall not repeat this point systematically. It is a property of the hyperbolic $3D$ space which we have to bear in mind while looking at the figures.

Note that in the figure, the entry 1 of an element is connected with the exit 4 of the previous one. Of

course, the segment can be run in the opposite direction: then an exit 4 becomes an entry 4 and an entry 1 becomes an exit 1.

In the righ-hand side picture of Figure 5, we represent another kind of track which we shall call **horizontal segments**. Such tracks consists of finitely many elements which can be written as a word of the form $(SeC)^k$, where Se denotes a straight element and C denotes a corner. The entry of the corner abuts an exit of the straight element. It is not always the same exit. In fact, there is an alternation of the exits which makes a Fibonacci word: if we associate to SeC the number of the exit of the straight element which abuts the entry of the corner, then this defines a homomorphism of $(SeC)^k$ on a factor of length k of the infinite Fibonacci word. Indeed, all corners are put on a black node of the Fibonacci tree. Straight elements are put on either white or black nodes. This can be made more accurate as follows. The straight elements of the segment are in contact of cells of the level n of the tree while the straight elements themselves are in the level n+1. The corners of the segment are all in the level n+2. Now, when the straight element is put on a white node, the exits are through faces 1 and 4. When it is put on a black node, the exits are through faces 1 and 10. This explains the connection of a horizontal segment with the infinite Fibonacci word, also see [6].

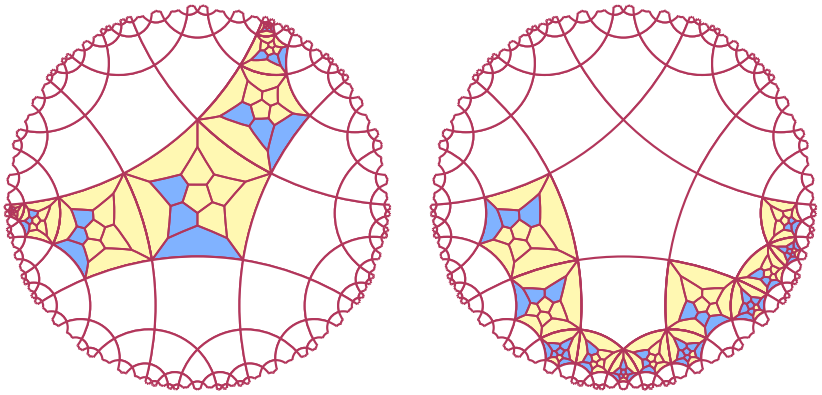


Figure 5 *Pseudo-projection on the plane of the track of its elements: left-hand side, case of a vertical segment; right-hans side, case of a horizontal one.*

In the right-hand side picture of Figure 5, all the cells, the leftmost one excepted, constitute an illustration of a horizontal segment. Note that the leftmost element does not belong to the horizontal segment but it realizes the connection with a vertical segment.

3.3 Bridges

As already mentioned in this section, crossings of the planar railway circuit are replaced by bridges. We can arrange the crossing in such a way that two vertical segments V_0 and V_1 cross each other. Assume that V_0 will remain in the plane Π_0 of its faces 5 while V_1 will make a detour in the plane Π_1, perpendicular to Π_0, which contains the guideline of its projection onto Π_0. In Π_1 the track will follow a horizontal segment which will take the cells of two circles of cells in Π_1: at a distance 2 or 3 from the cell c_0 of V_0 which has a contact with both Π_0 and Π_1. Figure 6 represent such a bridge using two pseudo-projections:

one onto the plane Π_0, on the left-hand side of the figure, and the other onto the plane Π_1 on its right-hand side. We shall say that the projection onto Π_0 is the view from above and that the projection onto Π_1 is the frontal view, both ways of views referring to the bridge itself.

Let us have a closer look at the figures.

In the view from above, we can see two vertical segments: one goes from the right-up part of the figure to the left-bottom one. It can be easily recognized as a copy of the vertical segment illustrated by Figure 5. Here, it contains two tiles coloured with light brown. We shall call this track the top-down track. The other track goes from the left-upper part of the figure and goes to the right-bottom one. We shall call it the left-right track. We can see the guideline of the top-down track. It is the intersection of the planes Π_0 and Π_1.

Still in the view from above, we can see golden yellow marks on the light brown tiles and two green marks on the central tile. The golden marks indicate that the top-down track goes on these tiles. The green marks indicate the two piles of the bridge, the light brwon tile being their basement. Number the cells of the projection of the top-down track in the view from above from 1 to 7, 1 being the number of the topmost cell. Cell 4 is the central cell and it belongs to the left-right track: the top-down track follows a horizontal segment on Π_1 which can be seen in the frontal view, see the right-hand side part of Figure 6. The departure/arrival of the horizontal segment is defined by cells 6 and 2 which can be seen on both views. In the frontal view, the trace of Π_0 can easily be seen: it is the border between the coloured tiles and the others which remain blank, on the bottom part of the figure. There are seven coloured cells along this line in the frontal view: they are exactly the cells number from 1 to 7 in the view from above. In the frontal view, cell 6 is on the left-hand side.

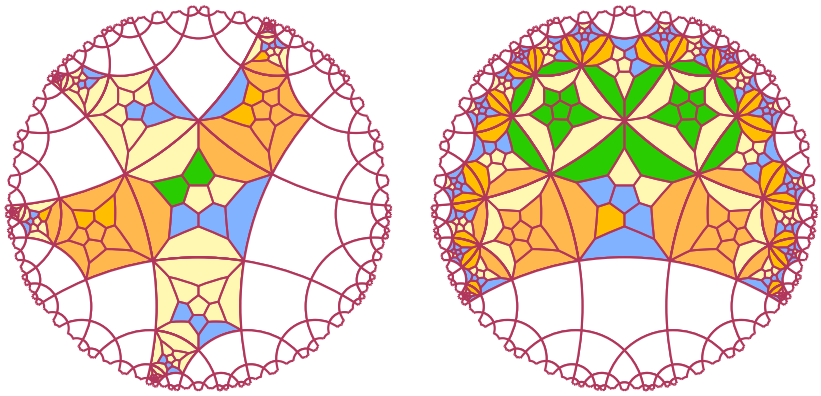


Figure 6 *Pseudo-projections of two tracks crossing through a bridge. On the left-hand side: pseudo-projection onto Π_0; on the right-hand side: pseudo-projection onto Π_1. The exit faces are marked by a golden yellow colour on the right-hand side figure. The piles of the bridge are marked with green on the central cell, their basement are marked with light brown. In the frontal view, the colour of the cell are exceptionally given to its faces which are not in contact with a cell of a track. This is to underline the elements of the bridge on which the track relies.*

Cells 6 and 2 are an application of what we have mentioned in Subsection 3.1, about the different ways to rotate a straight element in order to access to another plane. Cell 6 is alike the picture (c) of Figure 3. Its faces 0 and 5 are in contact with the guideline. Now, exit 3 may be used to go into the plane of face 0

which is perpendicular to the plane of face 5. This will be the starting point of our bridge. Note that face 0 of cell 6 is on Π_1. From this tile, the bridge follows a horizontal segment until it arrives at cell 2 which is in contact with both Π_0 and Π_1. Notice that cell 2 is alike the picture (e) of Figure 3, and that its face 4 is the face where the track of the bridges again joins the top-down track. Note that in cell 2, face 0 is on Π_1 as this is the case for cell 6. Looking at the cells of the horizontal segment in the frontal view, we can notice that the straight elements have a milestone which is below Π_1: this means that there are milestones on both half-spaces defined by Π_1. Now, for the corners, all the milestones are in a same half-space defined by Π_1. For the straight elements, their exits are most often the faces 1 and 4. However, from time to time, the exits are the faces 1 and 10. In Subsection 3.2 we have seen the reason of these variants. As can be seen on the frontal view, all corners are put on a black node of the Fibonacci tree and straight elements can be either on a black node or on a white one. From Subsection 3.2, we know that when a straight element is on a white tile, the exits are the faces 1 and 4. When it is on a black tile, it is the faces 1 and 10.

3.4 The motion of the locomotive on the tracks

Presently, we describe the motion of the locomotive on its tracks. We refer the reader to [11] for figures for all the possible motions on the tracks and through a switch.

This motion is very different from the simulation of [7]. There, the tracks were materialized by a specific colour and the locomotive simply occupied two contiguous cells of the track. Here, if we consider the track as constituted of the blank cells surrounded by milestones as in [9, 14, 13, 10], then the motion of the locomotive is very similar. In particular, it is exactly the same as the planar simulation described in [10]. Accordingly, restricting our attention to the cells of the track, we have the following one-dimensional rules for the motion of the locomotive:

```
B W W → B        W W B → B
R B W → R        W B R → R
W R B → W        B R W → W
W W R → W        R W W → W
```

As can easily be deduced from the rules, the locomotive consists of two contiguous cells: one is blue, the front, the other is red, the rear.

With the just mentioned principles in mind, we can easily device the rules for the motion of the locomotive. See [11] for the systematic writing of the corresponding rules and for illustrative figures.

From the rules we can devise, it is worth noticing that the elements can be freely assembled, provided that they observe the principle which we have fixed: exits of an element are 1 and 2 for corners, they are 1 and 3, 1 and 4, 1 and 8 or 1 and 10 for a straight element. Other combinations are ruled out by the rules.

4 Implementation of the switches

In order to describe the switches, we shall focus on the memory switch which has the most complex mechanism among the switches. In fact, this mechanism consists of two connected parts A and B. In the study of the other switches, we shall see that fixed switches use mechanism A alone and that the flip-flop switches use mechanism B alone.

All switches will share the following common features. They are assumed to be on the same plane Π_0. However, certain parts of the above mechanisms are on both half-spaces defined by Π_0. This is why we

shall present two figures for each switch: one is a pseudo-projection from above onto Π_0, the other is a pseudo-projection onto the same plane, but from below. We have to remember that in such a case, the left-hand side and the right-hand side are exchanged as well as clockwise and counter-clockwise motions.

Next, for two of them, the switches have both a left- and a right-hand side version. In the left-, right-hand side version respectively, the active passage sends the locomotive to the left-, right-hand side track respectively. However, for the fixed switch, a left-hand side version is enough. A right-hand side fixed switch is obtained from a left-hand side one as follows: after the switch, the left-hand side track crosses the right-hand side one in order to exchange the directions. Thanks to the bridge which we have implemented, this is easily performed.

The switches will be presented according to a similar scheme.

First, we describe what we call the **idle configuration**: it is the situation of the switch when it is not visited by the locomotive. All switches are the meeting point of three tracks. The meeting tile is a straight element and, in the figures, which will represent idle configurations only, it is placed at the central tile. The track which arrives to the entry 1 of this element represents the arrival for an active crossing of the switch. Exit 3 gives access to the track which goes to the left and exit 4 gives access to the track going to the right. In the computer program used to check the simulation, the cells of the tracks are numbered from 1 to 11 and from 12 to 16. In the figures, we can see cells 2 to 10 and 12 to 15 only. Cells 1 to 5 constitute the arriving track. They follow a vertical segment which arrives to the leading tile of a quarter constructed around the central cell. We shall number this sector by 1, as the exit to which the track leads. Cell 2 is the farthest visible cell from the central cell, cell 5 is the leading tile of sector 1. The central cell is cell 6. Cells 7 to 10 constitute the track which leaves the switch through exit 3. They are displayed in a vertical segment included in a sector lead by cell 7 and which is called sector 3, after exit 3. Cells 12 to 15 constitute the vertical segment which leaves the switch through exit 4. These cells belong to sector 4 headed by cell 12, see Figure 7 for instance.

A closer look shows that the tracks are not exactly along a vertical: the cell which is in contact with an exit of the central cell, is a straight element whose face 0 is on Π_0. The next cell, cell 4, 8 and 13 respectively is a corner, again with its face 0 on Π_0. The remaining two cells constitute a vertical segment in the way we have defined them with a milestone below Π_0 with respect to the other milestone which we consider as upon this plane.

With these conventions, we can start the study of each switch. We shall see the memory switches, the fixed switch and the flip-flop switches in this order.

4.1 Memory switches

As mentioned in the beginning of this section, the memory switches are the most complex construction in our implementation.

In the paper, we represent the left-hand side memory switch only, see Figure 7. The reader can see the figure corresponding to the right-hand side memory switch in [11].

In the figure, there is a big disc and a smaller one. The big disc is a pseudo-projection onto Π_0 from above, while the smaller one is a pseudo-projection onto the same plane from below. In both discs, we apply the convention about the colour of the cells.

In the memory switch, there are two **sensors**, two **markers** and two **controllers**. The sensors are cells 17 and 18 which are neighbours of the cells 7 and 12 respectively through their faces 0. Cells 7 and 12 are called the **scanned cells**, inspected by their sensors. The **upper** controller is cell 20 which is the second common neighbour of cells 7 and 12 above Π_0: the first common neighbour is the central cell.

We consider that cell 20 has its face 0 on Π_0. The **lower** controller is cell 19 which is the neighbour of cell 20 through its face 0: cell 19 is thus below Π_0 and we also consider that its face 0 is on Π_0. The two markers are cells 21 and 22: they are neighbours of cell 20 through its faces 8 and 10 respectively. Now, the sensor of cell 7 is blue and that of cell 12 is red. Similarly, cell 21 is red and cell 22 is blue. The colours of the sensors and of the markers allow to identify the left-hand side memory switch. In a right-hand side memory switch, the colours of the sensors and the markers are exchanged: cells 21 and 18 are blue, cells 22 and 17 are red.

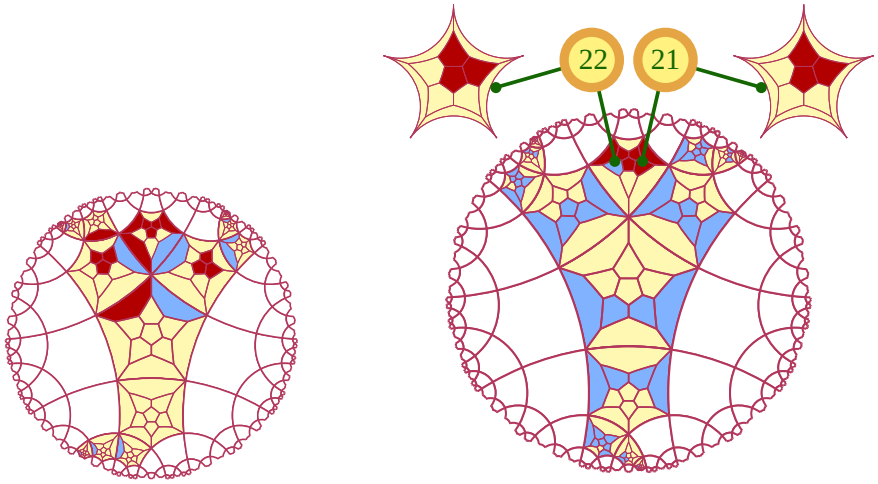


Figure 7 *The idle configuration of a left-hand side memory switch, represented by the two pseudo-projections, one from above: the big disc; the other from below: the small disc.*

The working of the memory switch is the following.

A blue sensor is indifferent to the direction of the locomotive: it may cross the cell it scans in both ways. A red sensor does not behave the same. First, it prevents the locomotive to enter the cell it scans in an active passage. In a passive passage, it detects the passage of the locomotive, it allows it to pass through the cell it scans, but it reacts to the passage by changing its colour: as the blue sensor does not see the red one, the red sensor cannot change its colour to blue. It changes it to white. This is detected by the lower controller, usually blue, which becomes red. When the lower controller is red, both sensors change their colour: the blue one to red the now white one to blue. And the lower controller goes back to blue. Now, the upper controller, usually blue, also detects the passive passage through the non-selected track: its markers allow it to differentiate cell 7 from cell 12. And so, when the front of the locomotive leaves cell 7 or cell 12 when this cell is on the non-selected track, the upper controller becomes white and then red. It becomes white to prevent the locomotive from being duplicated on the selected track: the locomotive must go through entry 1. Then it becomes red, at the same times as the lower controller becomes red. When the upper controller is red, both markers exchange their colour and at the next time, the upper controller returns to blue.

We have no room in this paper for figures about the motion of the locomotive. We refer the reader to [11] for such figures. In that document, the reader may also find tables of the execution of the computer

program which simulated the various motions. For the memory switch, we give one such table here: the table which corresponds to the motion of the locomotive when it passively crosses the switch from the non-selected track, see Table 1.

In a heading line, the trace indicates the visited cells by their number as well as their immediate neighbours, also numbered, when they may be changed during the visit. Below this leading line, for each time, the state of a cell is given in the column corresponding to its number. Looking at the numbers, we can see that the simulation program considered two more cells on the track arriving to the switch than what is shown by the figure. In this table, we can see that the sensors and the markers play an active role and, at the end of the role, they are exchanged. Accordingly, the locomotive entered a left-hand side memory switch and it leaves a right-hand side memory switch.

Note that Table 1 shows exactly when each sensor and controller is triggered. The front of the locomotive is in cell 12 at time 2. This makes cell 20 and 18 becoming white. As already noticed, the red sensor cannot change to blue as the blue sensor, which cannot see neither cell 12 nor cell 18, did not yet realized that a change must occur. At time 3, the front of the locomotive is now in cell 6, the central cell, and cells 20 and 18 are now white. This is the signal for both controllers to flash the red signal which will trigger the exchange of colours in the sensors and in the markers. The signal is sent at time 4 and the exchange of colours happens at time 5: starting from that time, the memory switch is now a right-hand side one.

Table 1 *Run of the simulation programme. A corresponding figure can be found in* [11]*. The passive crossing through the non-selected track correspond to cells* 12 *up to* 16*, in the reverse order and then to cells* 1 *up to* 6 *in the reverse order too.*

```
passive crossing of a memory switch, left-hand side,
  through the NON selected track :
         1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
time 0 :  W  W  W  W  W  W  W  W  W  W  W  W  W  B  R  W  B  R  B  B  R  B
time 1 :  W  W  W  W  W  W  W  W  W  W  W  W  B  R  W  W  B  R  B  B  R  B
time 2 :  W  W  W  W  W  W  W  W  W  W  W  B  R  W  W  W  B  R  B  B  R  B
time 3 :  W  W  W  W  W  B  W  W  W  W  W  R  W  W  W  W  B  W  B  W  R  B
time 4 :  W  W  W  W  B  R  W  W  W  W  W  W  W  W  W  W  B  W  R  R  R  B
time 5 :  W  W  W  B  R  W  W  W  W  W  W  W  W  W  W  W  R  B  B  B  B  R
time 6 :  W  W  B  R  W  W  W  W  W  W  W  W  W  W  W  W  R  B  B  B  B  R
time 7 :  W  B  R  W  W  W  W  W  W  W  W  W  W  W  W  W  R  B  B  B  B  R
```

In [11], it can be checked that the right-hand side memory switch reacts in a similar way to its passive crossing by the locomotive through the non-selected track. In particular, when the locomotive leaves the switch, it is now a left-hand side one.

4.2 Fixed switches

Figure 8 illustrates the idle configuration of a fixed switch. As announced at the beginning of Section 4 we can see on the figures that the idle configuration of a fixed switch is, in some sense the half of the configuration of a left-hand side memory switch. The point is that there is no lower controller and that the sensors and markers are now fixed milestones. Two of them, cell 18 and 21, are always red and the others, cell 17 and 22 are always blue. Now, the fixed switch keeps the upper controller. As already noticed, the red milestone prevents the locomotive to go through the non-selected track in an active passage. However, as also noticed in the study of memory switches, the change of colour in the sensor of cell 12 is not enough

to prevent the locomotive to go from the central cell both to cell 5, as required, and to cell 7 which should be avoided. Cell 7 cannot itself prevent such a passage because it sees cell 6 but it does not see at the same time cell 12. Now, we have seen in Subsection 4.1 that the upper controller is able to perform this tasks: as soon as it sees that the front of the locomotive is in cell 12, it becomes white. As cell 7 sees this new colour at the same time when the front of the locomotive is in cell 6, it allows it to reject the access of the locomotive to the selected track.

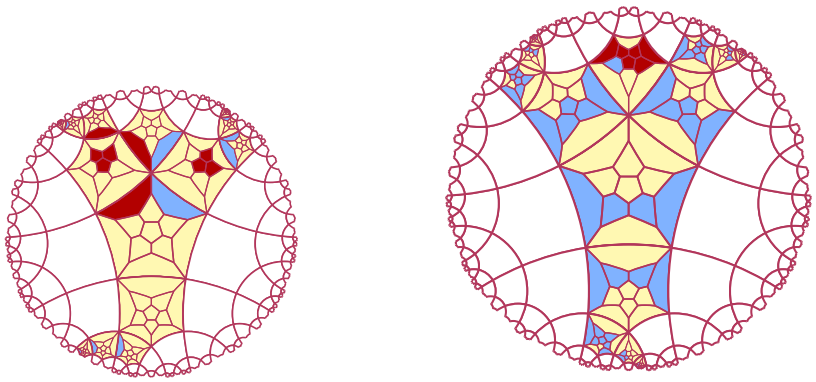


Figure 8 *The idle configuration of a fixed switch. The big disc is a view from above, the small one a view from below.*

It is not difficult to check that Figure 8 implements this changes. It was just enough to neutralize the lower controller by changing its colour from blue to blank. Moreover, the cell itself has no other non-blank neighbour than the sensors. Similarly, as the sensors and markers are fixed milestones, this means that their neighbours are all blank, except the cell of the switch with which they are in contact: cell 7 or 12 for the sensors, cell 20 for the markers. Consequently, the configuration of the fixed switch is a bit simpler than that of the left-hand side memory switch. It also requires less non-blank cells.

Table 2 *Run of the simulation programme corresponding to the passive crossing through the non selected track. The corresponding cells are cell* 12 *up to* 16*, in the reverse order and then cells* 1 *up to* 6 *in the reverse order too.*

```
passive crossing of a fixed switch, NON selected track :
          1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
time 0 :  W  W  W  W  W  W  W  W  W  W  W  W  B  R  W  B  R  W  B  R  B
time 1 :  W  W  W  W  W  W  W  W  W  W  W  B  R  W  W  B  R  W  B  R  B
time 2 :  W  W  W  W  W  W  W  W  W  W  B  R  W  W  W  B  R  W  B  R  B
time 3 :  W  W  W  W  W  B  W  W  W  W  R  W  W  W  W  B  R  W  W  R  B
time 4 :  W  W  W  W  B  R  W  W  W  W  W  W  W  W  W  B  R  W  R  R  B
time 5 :  W  W  W  B  R  W  W  W  W  W  W  W  W  W  W  B  R  W  B  R  B
time 6 :  W  W  B  R  W  W  W  W  W  W  W  W  W  W  W  B  R  W  B  R  B
time 7 :  W  B  R  W  W  W  W  W  W  W  W  W  W  W  W  B  R  W  B  R  B
```

We refer the reader to [11] for figures illustrating the three possible crossings of the switch by the locomotive. There, each figure is accompanied by a table which shows a trace of the execution of the

simulating program corresponding to that crossing. Here, we reproduce Table 2 only which gives the trace of a passive crossing through the non-selected track.

Table 2 also allows us to check that a half-control, namely that of cell 20 was enough to guarantee the correct working of the switch. It also shows that it was enough to block the changing of sensors by transforming them into milestones. This is an interesting point which shows us another advantage which we can take from the third dimension.

Indeed, in previous simulations in the hyperbolic plane on the heptagrid, with six or four states, we had a curious phenomenon during the active passage of the locomotive and also during a passive crossing for the fixed switch too. In these simulations, the passage of the locomotive created a duplicate of its front towards the wrong direction. However, as this new front was not followed by a red rear, it was possible to erase it, simply by appending a few rules.

4.3 Flip-flop switches

Figure 9 show the idle configuration of the left-hand side flip-flop switches. The corresponding figure for right-hand side switches can be seen in [11]. As announced at the beginning of Section 4 we can see on the figures that the idle configuration of a flip-flop switch switch is, in some sense the half of the configuration of a memory switch of the same laterality. The point is that there is no upper controller and, consequently, no markers. However, the sensors and the lower controller are still present. The lower controller is exactly the same as in the memory switches and it works in the same way. Contrarily to the fixed switch, the sensors are not milestones. They are true sensors like in the memory switch.

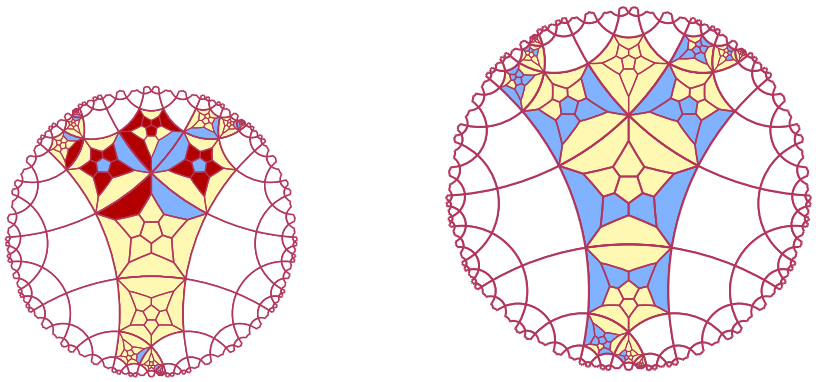


Figure 9 *The idle configuration of the left-hand side flip-flop switch. The big disc is a view from above, the small one, a view from below.*

However, they are a bit different from the sensors of the memory switch as they work in a different way. The difference can be noticed in the small discs of Figures 7 and Figures 9. In Figures 7, the sensors are marked by a group of three red milestones, on pairwise contiguous faces, one of them being the face which is opposite to that in contact with the scanned cell. In Figures 9, the sensors are marked by a ring of five milestones whose contact faces with the sensor are around the face which is opposite to the face in contact with the scanned cell. Also, the face opposite to that which is shared with the scanned cell is blue as it is in contact with a blue milestone. Similar figures for the right-hand side memory or flip-flop

switches can be found in [11].

Table 3 *Run of the simulation programme. A corresponding figure can be found in* [11]. *The active passage visits the cells of the selected track: cells* 1 *up to* 11 *in this order.*

```
active crossing of a left-hand side flip-flop switch :
          1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
time 0 :  W  R  B  W  W  W  W  W  W  W  W  W  W  W  W  W  B  R  B  W  W  W
time 1 :  W  W  R  B  W  W  W  W  W  W  W  W  W  W  W  W  B  R  B  W  W  W
time 2 :  W  W  W  R  B  W  W  W  W  W  W  W  W  W  W  W  B  R  B  W  W  W
time 3 :  W  W  W  W  R  B  W  W  W  W  W  W  W  W  W  W  B  R  B  W  W  W
time 4 :  W  W  W  W  W  R  B  W  W  W  W  W  W  W  W  W  B  R  B  W  W  W
time 5 :  W  W  W  W  W  W  R  B  W  W  W  W  W  W  W  W  W  R  B  W  W  W
time 6 :  W  W  W  W  W  W  W  R  B  W  W  W  W  W  W  W  W  R  R  W  W  W
time 7 :  W  W  W  W  W  W  W  W  R  B  W  W  W  W  W  W  R  B  B  W  W  W
```

This difference is explained by the fact that the working of the sensors is very different from that of the memory switch, quite the opposite: in the memory switch, the blue sensor is passive and the red sensor blocks the access to the non-selected track in the active passage and turns to white when the front of the locomotive appears in the scanned cell in a passive crossing. In the flip-flop switch, the red sensor only blocks the access to the non-selected track and the blue sensor is active: when the front of the locomotive leaves the scanned cell, it becomes white, triggering the flash of the lower controller at the next time which, to its turn, makes the sensors exchange their colour.

This can be checked in Table 3 and similar tables of [11]. The front of the locomotive is in the scanned cell at time 4, so that the blue sensor is white at time 5. As in the case of the memory switch, as cell 17 and 18 do not see each other, the blue sensor cannot turn to red immediately. It becomes white which triggers the flash of the controller at time 6 and the exchange of colours between the sensors at time 7 only.

5 About the rules and the computer program

We have no room for the rules which are displayed in [11]. However, we have a preliminary work on rotation invariance which is by itself interesting.

In order to write the rules of the cellular automaton, we shall use the numbering of the faces of a dodecahedron which was mentioned in Section 2 and which was used in Sections 3 and 4. However, there was no fixed rule to connect the numbering of a cell to that of a neighbouring one except for the cells of the track, as we did in Subsection 3.2. This is not a big problem as, in fact, the rules which we shall devise have an important property: they are **rotation invariant**, which means that they are not changed by a motion which leaves the dodecahedron globally invariant and which preserves orientation.

In the plane, the characterization of rotation invariance in the rules is easy to formulate: it is necessary and sufficient that the rules are not changed by a circular permutation on the neighbours. In the case of the pentagrid, this means that once we fixed a rule, we automatically append to the table of rules the other four permuted images of the rule. Here, the characterization is far less trivial. In [7], we could avoid this problem by imposing a stronger condition on the rules, namely to be **strongly lexicographically** different from each other. This means that to each rule, we associate a word of the form $A_1^{k_1}..A_n^n$ where A_1, ..., A_n

are the states and k_1 are non-negative numbers satisfying $k_1 + ... + k_n = v$+1, where v is the number of neighbours of the cell, the cell being not counted. This was possible with 5 states and I could not keep this condition for 3 states. This is why we first study how to check rotation invariance for our cellular automaton in the hyperbolic $3D$ space.

5.1 Rotation invariance

The question is the following: how does a motion which leaves the dodecahedron globally invariant affect the numbering of its faces, an initial numbering being fixed as in Section 2?

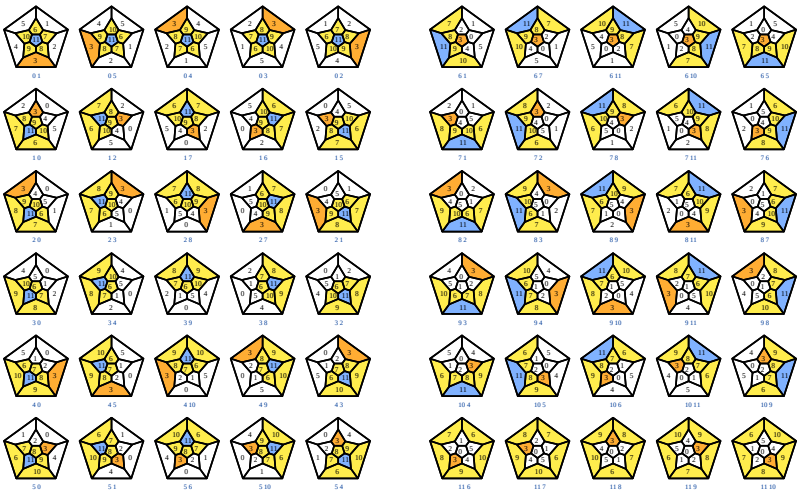


Figure 10 *The map of the positive motions leaving the dodecahedron globally invariant.*

Table 4 *The faces around a given face.*

	1	2	3	4	5
0	1	5	4	3	2
1	0	2	7	6	5
2	0	3	8	7	1
3	0	4	9	8	2
4	0	5	10	9	3
5	0	1	6	10	4
6	1	7	11	10	5
7	1	2	8	11	6
8	2	3	9	11	7
9	3	4	10	11	8
10	4	5	6	11	9
11	6	7	8	9	10

In fact, it is enough to consider motions which preserves the orientation, we shall say **positive** motions. As such a motion leaves the dodecahedron globally invariant, it transforms a face into another one. Accordingly, fix face 0. Then its image, say f_0, can be any face, face 0 included. Next, fix a second face which shares an edge with face 0, for instance face 1. Then its image, say f_1, is a face which shares an edge with f_0. It can be any face sharing a face with f_0. Indeed, let f_2 be another face sharing an edge with f_1. Then, composing the considered positive motion with a rotation around f_0 transforming f_1 into f_2, we get a positive motion which transforms $(0, 1)$ into (f_0, f_2). This proves that we get all positive motions leaving the dodecahedron globally invariant, by first fixing the image f_0 of face 0 and then by taking any face f_1 sharing an edge with f_0. Note that once f_0 and f_1 are fixed, the images of the other faces are fixed, thanks to the preservation of the orientation. Accordingly, there are 60 of these positive motions and the argument of the proof shows that they are all products of rotations leaving the dodecahedron globally invariant.

Figure 10 gives an illustrative classification of all these rotations. The upper left picture represents the image of a Schlegel diagram of a dodecahedron with the notation introduced in Section 2. Each image represents a positive motion characterized by the couple of numbers under the image: it has the form $f_0\ f_1$, where f_0 is the image of face 0 and f_1 is the image of face 1. The figure represents two sub-tables, each one containing 30 images. Each row represents the possible images of f_1, f_0 being fixed. The image of face 0 is the plane of projection of the dodecahedron. The image of face 1 takes the place of face 1 in Figure 1. As an example, $f_0 = 0$ for the first row of the left-hand side sub-table, and in the first row, the first image gives $f_1 = 1$, so that it represents the identity. The other images of the row represent the rotations around face 0.

The construction of Figure 10 was performed by an algorithm using Table 4. For each face of the dodecahedron, the table gives the faces which surround it in the Schlegel diagram, taking the clockwise order when looking at the face from outside the dodecahedron, this order coinciding with increasing indices in each row. This coincides with the usual clockwise order for all faces as in Figure 1, except for face 0 for which the order is counter-clockwise when looking above the plane of the projected image. The principle of the drawings consists in placing f_0 onto face 0 and f_1 onto face 1. The new numbers of the faces are computed by the algorithm as follows. Being given the new numbers f_0 and f_1 of two contiguous faces φ_0 and φ_1 in the Schlegel diagram, the algorithm computes the position of φ_1 as a neighbour of φ_0 in the table. This allows to place f_1 on the right face. Then, the algorithm computes the new numbers of the faces which are around φ_1 in the table: it is enough to take the position of φ_0 as a neighbour of φ_1 and then to turn around the neighbours of f_1, looking at the new numbers in the row f_1 of the table, starting from the position of f_0. This gives the new numbers of the faces which surround face 1. It is easy to see that we have all faces of the dodecahedron by turning around face 1, then around face 5, then around face 7 and at last around face 8. As in these steps, each round of faces starts from a face whose new number is already computed, the algorithm is able to compute the new numbers for the current round of faces, using Table 4 to find the new numbers. Let us call this algorithm the **rotation algorithm**.

Thanks to the rotation algorithm, it is easy to compute the **rotated forms** of a rule of the cellular automaton.

Let $\underline{\eta}\eta_0 ... \eta_{11}\underline{\eta'}$ be a rule of the automaton. In this format, η is the current state of the cell and $\eta(i)$ is the state of the neighbour through face i, also called neighbour i, and η' is the new state of the automaton. Remember that the current state of a cell is its sate at time t and that its new state is its state at time t+1. Call $\underline{\eta}\eta_0 ... \eta_{11}$ the **context** of the rule. Let μ be a positive motion leaving the dodecahedron globally invariant. The **rotated form** of the rule defined by μ is $\underline{\eta}\eta_{\mu(0)} ... \eta_{\mu(11)}\underline{\eta'}$ and, similarly, $\underline{\eta}\eta_{\mu(0)} ... \eta_{\mu(11)}$ is

the **rotated form** by μ of the context of the initial rule. We say that the cellular automaton is **rotation invariant** if and only two rules having contexts which are rotated forms of each other always produce the same new state.

Now, thanks to our study, we have a syntactic criterion to check this property. We fix an order of the states. Then, for each rule, we compute its **minimal form**. This form is obtained as follows. We compute all rotated forms of the rule and, looking at the obtained contexts as words, we take their minimum in the lexicographic order. The minimal form of a rule is obtained by appending its new state to this minimum. Now it is easy to see that:

Lemma 1 *A cellular automaton on the dodecagrid is rotation invariant if and only if for any pair of rules, if their minimal forms have the same context, they have the same new state too.*

Now, checking this property can easily be performed thanks to the rotation algorithm.

We refer to [11] for the detailed study of the rules. The rules given there have the property that the minimal forms of their contexts are pairwise distinct.

5.2 About the computer program

As indicated in the introduction, I wrote a computer program in order to check the correctness of the rules. The program was written in $ADA95$ and it implements the algorithms mentioned in the paper.

Before giving a short account on the program itself, I would like to stress that using a simulation program for this purpose is mandatory. The computations are so complex for a man, at least for me, that the help of the computer allows me to check that the rules are correct. What is meant by this latter expression? We mean two things: a syntactical one and a semantic one. The syntactical correctness is that there is no pair of rules with the same minimal context giving rise to different new states. This is the minimal condition when working with deterministic cellular automata, which is of course the case here. In this work, we reinforced the condition by checking that two rules with the same minimal context always give rise to the same new state: this guarantees that the automaton is not only correct, but that it is also rotation invariant.

Now, the semantic correctness means that the rules do what we expect from them to do. This is far more complex to check and this cannot be completely ascertained by proof. Again, the computer program is useful in this regard. We can implement the simulation in the program and then run it. If everything goes smoothly through, we can believe that the implementation is correct. There is no guarantee of that. There is no automatic checking that the implementation is a correct hyperbolic implementation. There is also no proof that the program itself is correct. However, the setting is rather involved and while adjusting the program, many errors in my first table of rules were found by the program. As an example, the program also indicated me the need to cover face 11 of cells 7 and 12 with a blue milestone: otherwise, the set of rules would not be rotationally invariant.

About the program implementation itself.

First, I implemented the algorithms to compute the minimal form of a rule and, taking advantage of this implementation, the programme computed the PostScript program for Figure 10. All traces given in the tables of Section 4 were computed during the execution of the program by the program itself.

Each test of a crossing of the switch by the locomotive was performed within the same implementation frame: the cells of the tracks were gathered in a table of tables. The big table has 22 entries corresponding to the numbering of the cells explained in Section 4. For each index of the big table, a table of 14 entries gives various information on the cell in its current state, and its neighbours. The neighbours correspond

to faces and are numbered from 0 up to 11 as in the paper. For each face, it is indicated whether the neighbour through the face has a permanent state or a variable one. As an example, a milestone seen from a face of a cell of the track is permanently blue. When the neighbour has a variable state, the table indicates the index of this neighbour in the big table. In fact, the big table is first a list of the variable cells and, at this occasion, it collects a useful information about each cell. The program also computes a bigger trace where at each time, the big table is dispatched in full detail. Table 5 gives two short pieces of this trace, taken during an active passage of the locomotive, at time 1. We can see the information which was just indicated, v meaning 'variable' and f meaning 'fixed'. The other indications are self-explaining.

Table 5 *Two pieces of the big trace of the program, corresponding to the simulation of an active passage of the locomotive through a left-hand side memory switch, at initial time.*

```
 6   -1  0  1  2  3  4  5  6  7  8  9 10 11
      W  W  W  B  W  W  B  B  B  W  W  W  W
      v  f  v  f  v  v  f  f  f  f  f  f  f
            5     7 12
 7    W  B  W  B  W  W  B  B  B  W  W  W  B
      v  v  v  f  f  v  v  f  f  f  f  f  f
        17  6        8 20
 8    W  W  W  B  W  W  B  B  B  W  W  W  W
      v  f  v  f  f  v  f  f  f  f  f  f  f
            7        9
 9    W  W  W  B  W  W  B  B  B  W  W  W  W
      v  f  v  f  f  v  f  f  f  f  f  f  f
            8       10
10    W  W  W  B  W  W  B  B  B  W  W  W  W
      v  f  v  f  f  v  f  f  f  f  f  f  f
            9       11
11   -1  0  1  2  3  4  5  6  7  8  9 10 11
      W  W  W  B  W  W  B  B  B  W  W  W  W
      v  f  v  f  f  f  f  f  f  f  f  f  f
           10
12    W  R  W  B  W  W  B  B  B  W  W  W  B
      v  v  v  v  f  v  f  f  f  f  f  f  f
        18  6 20    13
13    W  W  W  B  W  W  B  B  B  W  W  W  W
      v  f  v  f  f  v  f  f  f  f  f  f  f
           12       14
```

```
17    B  W  W  B  W  W  W  W  W  W  R  R  R
      f  v  f  v  f  f  f  f  f  f  f  f  f
         7    19
18    R  W  W  W  W  W  B  W  W  R  R  W  R
      f  v  f  f  f  f  v  f  f  f  f  f  f
        12           19
19    B  B  W  R  B  R  R  R  R  W  W  W  R
      f  v  f  f  v  v  f  f  f  f  f  f  f
        20       17 18
20    B  B  W  R  W  W  R  R  R  R  W  B  R
      f  v  f  f  v  v  f  f  f  v  f  v  f
        19       12  7          21    22
21   -1  0  1  2  3  4  5  6  7  8  9 10 11
      R  B  W  W  W  W  W  W  W  W  R  R  R
      v  v  f  f  f  f  f  f  f  f  f  f  f
        20
22    B  B  W  W  W  W  W  W  W  W  R  R  R
      v  v  f  f  f  f  f  f  f  f  f  f  f
        20
```

As the program performed a successful execution of all possible crossings and also along various vertical and horizontal segments with some mix of them, we can conclude that the proof of theorem 1 is complete. ■

References

[1] M. Cook. Universality in elementary cellular automata, *Complex Systems*, (2004), **15**(1), 1-40.

[2] F. Herrmann, M. Margenstern, A universal cellular automaton in the hyperbolic plane, *Theoretical Computer Science*, (2003), **296**, 327-364.

[3] M. Margenstern, Implementing Cellular Automata on the Triangular Grids of the Hyperbolic Plane for New Simulation Tools, **ASTC'2003**, (2003), Orlando, March, 29- April, 4.

[4] M. Margenstern, The tiling of the hyperbolic $4D$ space by the 120-cell is combinatoric, *Journal of Universal Computer Science*, **10**(9), (2004), 1212-1238.

[5] M. Margenstern, Two railway circuits: a universal circuit and an NP-difficult one, *Computer Science Journal of Moldova*, **9**, 1–35, (2001).

[6] M. Margenstern, Tilings of hyperbolic spaces: the splitting method and group theory, **WORDS'2003**, TUCS General Publications, **43**, (2003), 31-35.

[7] M. Margenstern, A universal cellular automaton with five states in the 3D hyperbolic space, *Journal of Cellular Automata* **1**(4), (2006), 315-351.

[8] M. Margenstern, Cellular Automata in Hyperbolic Spaces, Volume 1, Theory, *OCP*, Philadelphia, (2007), 422p.

[9] M. Margenstern, Cellular Automata in Hyperbolic Spaces, Volume 2, Implementation and computations, *OCP*, Philadelphia, (2008), 360p.

[10] M. Margenstern, A universal cellula automaton on the heptagrid of the hyperbolic plane with four states, *Theoretical Computer Science*, (2010), *doi:*10.1016/j.tcs.2010.04.015.

[11] M. Margenstern, A weakly universal cellular automaton in the hyperbolic $3D$ space with three states, *arXiv*:1002.4290[cs.DM], (2010), 54pp.

[12] M. Margenstern, G. Skordev, Tools for devising cellular automata in the hyperbolic 3D space, *Fundamenta Informaticae*, **58**, N°2, (2003), 369-398.

[13] M. Margenstern, Y. Song, A universal cellular automaton on the ternary heptagrid, *Electronic Notes in Theoretical Computer Science*, **223**, (2008), 167-185.

[14] M. Margenstern, Y. Song, A new universal cellular automaton on the pentagrid, *Parallel Processing Letters*, **19**(2), (2009), 227-246.

[15] D.M.Y. Sommerville, An introduction to the geometry of N dimensions, Dover Publ. Inc., New-York, 1958.

[16] I. Stewart, A Subway Named Turing, Mathematical Recreations in *Scientific American*, (1994), 90-92.

[17] S. Wolfram. A new kind of science, *Wolfram Media, Inc.*, (2002).

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 115–130

Minimal Recurrent Configurations of Chip Firing Games and Directed Acyclic Graphs

Matthias Schulz[1]

[1] *Karlsruhe Institute for Technology,*
Department for Computer Sciences
Am Fasanengarten 5, 76128 Karlsruhe, Germany
schulz@ira.uka.de

We discuss a very close relation between minimal recurrent configurations of Chip Firing Games and Directed Acyclic Graphs and demonstrate the usefulness of this relation by giving a lower bound for the number of minimal recurrent configurations of the Abelian Sandpile Model as well as a lower bound for the number of firings which are caused by the addition of two recurrent configurations on particular graphs.

Keywords: Chip Firing Games, Sandpile Model, Minimal Recurrent Configurations, DAGs, Addition of Recurrent Configurations

1 Introduction

The Abelian Sandpile Model was introduced by Bak, Tang and Wiesenfeld in 1987 [1] as a model to explain $\frac{1}{f}$ noise. We assign each point of a $n \times n$ grid a number of grains of sand, then taking points which contain at least four grains of sand and let one grain topple to each of the four adjacent points; if a point on the edge of the grid is chosen, grains fall out of the system.

This dynamic is closely related to Chip Firing Games, and Chung and Ellis proposed a variation of Chip Firing Games in 2002 [3] such that the Abelian Sandpile Model can be seen as a special case of this model.

Dhar found many nice properties of so-called recurrent configurations of the Abelian Sandpile Model which are together with a natural operation $\oplus$ an Abelian group, see [5]. These findings can be generalized for Chung and Ellis' Chip Firing Game, as shown in [3].

Recurrent configurations of the Abelian Sandpile Model or Chip Firing Games are characterized by containing enough grains of sand/chips; in this paper we will look at configurations which contain as few chips as possible for a recurrent configuration, and are able to prove a close relation to directed acyclic graphs (DAGs). This relation somewhat resembles the bijection between the set of recurrent configurations of the Sandpile Model and the set of spanning trees with roots at the border of the grid which was shown in [8] and generalized for Chip Firing Games in [3].

These recurrent configurations which we will call minimal play a significant part when considering minimization problems on the set of recurrent configurations, most naturally when minimizing the nuber of firings that occur when relaxing the sum of two recurrent configurations as in [9].

First, we will introduce the basic concepts for Chip Firing Games, before examining the relation between minimal recurrent configurations and a subset of the DAGs on the graph underlying the Chip Firing Game. We can use this to prove a lower bound for the number of minimal recurrent configurations of the Abelian Sandpile Model.

Then we will define a dynamic on DAGs which corresponds to the dynamic of the Chip Firing Game. Using this correspondence we will be able to give the infimum of the number of firings that occur when we start the Chip Firing Game on a cylindrical grid with the sum of two recurrent configurations.

2 Preliminaries

2.1 Basic Definitions

An undirected graph $U = (V \cup S, E)$ is called a *CFG-graph* iff V and S are disjoint, each vertex $s \in S$ is adjacent to exactly one vertex $v \in V$ and there exists a path from each vertex $v \in V$ to a vertex $s \in S$.

A Chip Firing Game (CFG) on a CFG-graph defines a transition rule for configurations $c : V \to \mathbb{N}_0$ where we interpret $c(v)$ as the number of chips the vertex v contains:

If a vertex $v \in V$ contains at least $\deg(v)$ chips, where $\deg(v)$ is the degree of v in the Graph U, the vertex v is called *critical* and can fire, *i.e.* give one chip to each adjacent vertex and lose $\deg(v)$ chips.

Chips which are given to vertices in S simply vanish from the game. Figure 1 gives an example; black vertices stand for vertices in S.

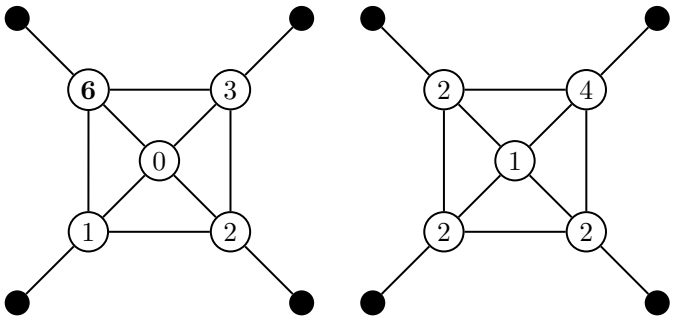


Fig. 1: The vertex in the upper left corner fires, and we get the configuration on the right.

If we start with a configuration c and get configuration c' after vertex $v \in V$ has fired, we write $c' = \phi_v(c)$. We can write

$$\phi_v(c) = c - \deg(v)e_v + \sum_{v' | \{v,v'\} \in E} e_{v'},$$

e_v being the configuration given through $\forall u \in V : e_v(u) = \delta_v(u)$.

A configuration which contains a vertex which is able to fire is called *critical*; a configuration which is not critical is called *stable*, and the set of stable configurations is denoted $\mathcal{C}^U$.

2.2 Relaxations of Configurations

It has been shown (for example in [3]) that we reach a stable configuration after a finite number of firings, no matter which critical configuration we start from. We call the process of these firings the *relaxation of* c.

For $k \in \mathbb{N}_0$ listing the vertices which fired during the first k steps of the relaxation of c is called a *firing sequence* of c of length k.

It is also shown in [3] and [5] that the stable configuration reached does not depend on the sequence of firings - there exists a unique stable configuration c_{rel} we reach when starting with configuration c, and even the number of times a given vertex v fires during the relaxation is unique. The vector f_c assigning each vertex the number of times it fires during the relaxation of c is called the *firing vector* of c.

Throughout this paper, when comparing different firing vectors or different configurations, we will use the relation $\leq$ defined through $c \leq d \iff \forall v \in V : c(v) \leq d(v)$.

2.3 The Operation $\oplus$ and Recurrent Configurations

Definition 1 *We define the operation $\oplus$ on $\mathcal{C}^U$ through*

$$\forall c, d \in \mathcal{C}^U : c \oplus d = (c + d)_{rel}. \tag{1}$$

(The operation $+$ is the usual pointwise addition of functions.)

It is shown in [3] that $\oplus$ is commutative and associative, and also that there exists a subset of stable configurations $\mathcal{R}^U$ such that $(\mathcal{R}^U, \oplus)$ is an Abelian group. These configurations are called *recurrent configurations*.

The structure of the Abelian group $(\mathcal{R}^U, \oplus)$, called the *Sandpile Group* of the graph U, has been the object of research for U being a complete graph or an n-wheel in [4] or U being a tree in [7]. Furthermore the geometrical structure of the neutral element of said group has been discussed in [2].

Definition 2 *We define $b \in \mathcal{C}^U$ as the configuration which assigns to each vertex v the number of vertices in S which are adjacent to v. The configuration b is called the* burning configuration of U.

A generalization of Dhar's Burning Algorithm from [8] gives us the following equivalence:

$\forall c \in \mathcal{C}^U : c \in \mathcal{R}^U \iff$ there exists a firing sequence for $c + b$ which contains each vertex exactly once.

(Note that for all $c \in \mathcal{C}^U$ the firing sequence of $c + b$ contains each vertex at most once.)

For the rest of this paper, we will say that a sequence F of vertices is a firing sequence for a recurrent configuration c if F is a firing sequence for $c + b$ of length $|V|$.

We are now able to proceed to the actual subject matter of this paper, the set of minimal recurrent configurations.

3 Minimal Recurrent Configurations and Firing Graphs

Definition 3 *A recurrent configuration $c \in \mathcal{R}^U$ is called* minimal recurrent *if for all vertices $v \in V$ satisfying $c(v) > 0$ the configuration $c - e_v$ is not recurrent.*

The set of all minimal recurrent configurations on U shall be denoted $\mathcal{R}^U_{min}$.

In other words $\mathcal{R}^U_{min}$ is the set of minimal elements in $\mathcal{R}^U$ with regard to the partial order $\leq$ as defined above.

These minimal recurrent configurations occur naturally when one tries to lower the number of firings that happen during the relaxation of the sum of two recurrent configurations: The function $f : \mathbb{N}_0^V \to mathbbN_0^V, c \mapsto f_c$ is monotonously increasing with regards to $\leq$; this means that $f_{c+d} \leq f_{c'+d'}$ if $c \leq c'$ and $d \leq d'$ is true, and we get minimal results for some minimal recurrent configurations c, d.

To get a better understanding of minimal recurrent configurations, we use *firing graphs*, a concept also used by Gajardo and Goles in [6]:

Definition 4 *Let $c \in \mathcal{R}^U$ be a recurrent configuration and $F = (v_0, \ldots, v_{|V|-1})$ a firing sequence for c.*

We define the firing graph $G_F = (V \cup S, E')$ *by choosing*

$$E' = \{(v_i, v_j) \mid \{v_i, v_j\} \in E \wedge i < j\} \cup \{(s, u) \mid s \in S \wedge \{s, u\} \in E\} \tag{2}$$

We also say that G_F is a firing graph for c.

Example: The configuration given in Figure 2 has the firing sequences $F = (0, 1, 2, 3, 4)$ and $F' = (1, 0, 2, 3, 4)$. The resulting firing graphs G_F and $G_{F'}$ are shown in Figure 3.

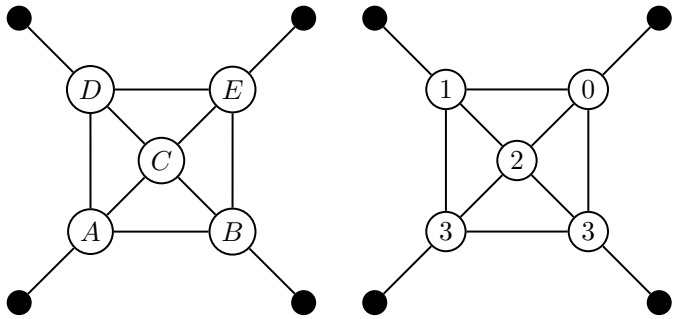


Fig. 2: A CFG-Graph U and a recurrent configuration on U; the black vertices are the vertices in S.

Note that F is always a topological ordering of G_F restricted to V, which implies that G_F is always a directed acyclic graph. Note also that for each edge $\{u, v\} \in E$ either (u, v) or (v, u) is an edge in G; we will call such acyclic graphs *DAGs on U* and *S-DAGs on U* if S is the set of sources of G.

The set of all S-DAGs on U shall be denoted $\mathcal{D}^U_S$.

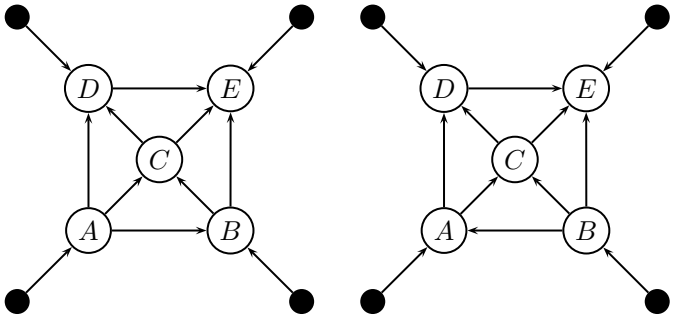


Fig. 3: The firing graphs for the firing sequences (A, B, C, D, E) (left) and (B, A, C, D, E) (right).

Definition 5 *As we will be discussing indegrees and outdegrees of vertices in different graphs, we define for a directed graph $G = (V \cup S, E')$ the functions*

$$indeg_G : V \cup S \to \mathbb{N}_0, \qquad v \mapsto |\{u \in V \cup S \mid (u, v) \in E'\}| \tag{3}$$

$$outdeg_G : V \cup S \to \mathbb{N}_0, \qquad v \mapsto |\{u \in V \cup S \mid (v, u) \in E'\}|. \tag{4}$$

Note that for all $v \in V$ and all DAGs G on U the equation $indeg_G(v) + outdeg_G(v) = \deg(v)$ is true.

Lemma 1 *Let $c \in \mathcal{R}^U$ be a recurrent configuration and G be an S-DAG on U.*
Then G is a firing graph of c iff for all vertices $v \in V$ the statement $outdeg_G(v) \leq c(v)$ is true.

Proof: If G is a firing graph of c there exists a firing sequence F of c such that $G = G_F$ is true.

It follows that the number of chips fallen to a vertex v before it fires is the number of neighbors firing before it in the firing sequence F plus the numbers of neighbors v has in S. These are exactly the vertices from which an edge goes to v in G.

As v has enough chips to fire after the chips of the neighbors mentioned above have fired, it follows $c(v) + indeg_G(v) \geq \deg(v)$, which leads to $c(v) \geq outdeg_G(v)$, which proves one direction.

For the other direction, assume that $\forall v \in V : c(v) \geq outdeg_G(v)$ is true. Let $F = (v_0, \ldots, v_{|V|-1})$ be a topological ordering of G restricted to V.

We show that F is a firing sequence for $c + b$:

We define for $0 \leq i \leq |V|$ the configuration $c_i : V \to \mathbb{Z}$ as

$$c_0 = c + b, \forall i \in \{0, \ldots, |V| - 1\} : c_{i+1} = \phi_{v_i}(c_i) \tag{5}$$

In other words, c_i is the configuration we get after the first i vertices of the sequence have fired.

To show that F is indeed a firing sequence, we have to prove that for all i the inequation $c_i(v_i) \geq \deg(v_i)$ is true.

The number of chips v_i contains in c_i is the sum of the number of chips v_i contained in c, the number of neighbors v_i has in S and the number of neighbors which fired before v_i in F; in other words

$$c_i(v_i) = c(v_i) + indeg_G(v_i) \geq outdeg_G(v_i) + indeg_G(v_i) = \deg(v_i) \tag{6}$$

This completes the proof.

□

Note that we didn't use the fact that c has to be a recurrent configuration for the second part of the proof; indeed, we have shown we can find a firing sequence for $c + b$ comprising all vertices of V exactly once if $\forall v \in V : c(v) \geq outdeg_G(v)$ is true.

This means that all configurations $c \in \mathcal{C}$ satisfying $\forall v \in V : c(v) \geq outdeg_G(v)$ are recurrent, a fact we state in the following lemma:

Lemma 2 *Let G be an S-DAG on U and $c \in \mathcal{C}^U$ a configuration satisfying $\forall v \in V : c(v) \geq outdeg_G(v)$. Then c is a recurrent configuration and G is a firing graph of c.*

As the sources of G are exactly the vertices in S this means that for all $v \in V$ the inequation $outdeg_G(v) \leq \deg(v) - 1$ holds. Therefore such a configuration c always exists in $\mathcal{C}$.

We also can use the firing graphs to prove a lower bound for the number of chips a recurrent configuration contains:

Corollary 1 *We define $E_V \subseteq E$ as the set of all edges in U incident to two vertices in V. Then the following inequation holds:*

$$\forall c \in \mathcal{R}^U : \sum_{v \in V} c(v) \geq |E_V| \tag{7}$$

Proof: Let $c \in \mathcal{R}^U$ be a recurrent configuration and $G = (V \cup S, E')$ be a firing graph of c. Using Lemma 2, we get

$$\forall c \in \mathcal{R}^U : \sum_{v \in V} c(v) \geq \sum_{v \in V} outdeg_G(v) \tag{8}$$

Therefore c contains at least as many chips as there are edges in G starting from a vertex $v \in V$. As each edge $\{u, v\} \in E_V$ satisfies $(u, v) \in E' \vee (v, u) \in E'$ and no edge in G goes from a vertex $v \in V$ to a vertex $s \in S$, we get $\sum_{v \in V} outdeg_G(v) = |E_V|$ which proves the claim.

□

We know that in a recurrent configuration c with firing graph G each vertex contains at least $outdeg_G(v)$ chips. Looking at configurations where each vertex v contains exactly $outdeg_G(v)$ chips leads us to the following theorem:

Theorem 1 *A configuration $c \in \mathcal{C}^U$ is minimal recurrent iff there is an S-DAG G such that $\forall v \in V : c(v) = outdeg_G(v)$ is true.*

Proof: If an S-DAG G exists such that $\forall v \in V : c(v) = outdeg_G(v)$ is true, Lemma 2 tells us that c is a recurrent configuration and G is a firing graph of c.

Looking at the proof of Corollary 1 we also get $\sum_{v \in V} c(v) = \sum_{v \in V} outdeg_G(v) = |E_V|$, which is the smallest number of chips a recurrent configuration on U can contain.

For all vertices $v \in V$ the configuration $c - e_v$ contains fewer than $|E_V|$ chips, so none of these configurations can be recurrent.

Therefore c is a minimal recurrent configuration.

Now, let c be a minimal recurrent configuration and G a firing graph of c. We know that $\forall v \in V : c(v) \geq outdeg_G(v)$ is true.

We define $c' \in \mathcal{R}^U$ as the configuration satisfying $\forall v \in V : c'(v) = outdeg_G(v)$.

We get $c \geq c'$; as c is a minimal recurrent configuration, c' must be the same configuration as c, which completes the proof.

□

As we found that the S-DAG G claimed to exist for minimal recurrent configuration C in Theorem 1 is a firing graph of c, we get the following corollary:

Corollary 2 *If c is a minimal recurrent configuration and G is a firing graph of c, $\forall v \in V : c(v) = outdeg_G(v)$ is true.*

We have shown that there exists a relation between S-DAGs on U and minimal recurrent configurations on U. We now show that we can even find a bijection between the set of minimal recurrent configurations on U and the set of S-DAGs on U.

To do so, we have to show that no minimal recurrent configuration has more than one firing graph, which is shown in the following theorem:

Theorem 2 *A minimal recurrent configuration has only one firing graph.*

Proof: Suppose the minimal recurrent configuration c has two different firing graphs $G_1 = (V \cup S, E_1)$ and $G_2 = (V \cup S, E_2)$.

Then we get $\forall v \in V : outdeg_{G_1}(v) = c(v) = outdeg_{G_2}(v)$ according to Corollary 2.

Consider the set $C = \{(u, v) \in E_1 \mid (v, u) \in E_2\}$ of edges which are in G_1 but not in G_2 and the set $V_C = \{v \in V \mid \exists u \in V : (u, v) \in C \vee (v, u) \in C\}$ of vertices incident to edges in C.

Then the graph $G_C = (V_C, C)$ is a subgraph of G_1 and as such a DAG, which means it that it contains a sink u with outdegree zero and a vertex v such that $(v, u) \in C$.

As u is a sink in G_C, each vertex u' satisfying $(u, u') \in E_1$ also satisfies $(u, u') \in E_2$, as the edge (u, u') would otherwise be contained in C.

We also know that $(u, v) \in E_2$, as $(v, u) \in C$. Therefore the outdegree of u in G_2 is at least one higher than the outdegree of u in G_1, which is a contradiction.

Therefore c can have no two different firing graphs.

□

Theorem 2 shows that we can easily assign an S-DAG G to each minimal recurrent configuration c by choosing G as the unique firing graph of c. We now show that this function is a bijection, which we will afterwards use to prove a lower bound on the number of minimal recurrent configurations if the underlying graph U is a grid.

Definition 6 *We define $\tau : \mathcal{R}^U_{min} \to \mathcal{D}^U_S$ as the function which is defined through*

$$\forall c \in \mathcal{R}^U_{min} : \tau(c) \text{ is the firing graph of} c.$$

We also define the function $\rho : \mathcal{D}^U_S \to \mathcal{R}^U_{min}$ through

$$\forall G \in \mathcal{D}^U_S : \forall v \in V : (\rho(G))(v) = outdeg_G(v)$$

Corollary 3 *The functions τ and ρ are inverse functions and therefore bijections.*

Proof:

Let $c \in \mathcal{R}^U_{min}$ be a minimal recurrent configuration. We know $\forall v \in V : c(v) = outdeg_{(\tau(c))(v)} = (\rho(\tau(c)))(v)$ according to Corollary 2, which means that $\rho \circ \tau$ is the identity on $\mathcal{R}^U_{min}$.

Let $G \in \mathcal{D}^U_S$ be an S-DAG of U. The configuration $\rho(G)$ is a minimal recurrent configuration according to Lemma 1. As G is a firing graph of $\rho(G)$ according to Lemma 1 and $\rho(G)$ has only one firing graph according to Theorem 2, it follows that $\tau(\rho(G)) = G$, and therefore $\tau \circ \rho$ is the identity on $\mathcal{D}^U_S$.

Therefore $\rho = \tau^{-1}$, which means that τ and ρ are bijections.

□

We will use this bijection to prove a lower bound for the number of minimal recurrent configurations of the Abelian Sandpile Model:

Lemma 3 *Let $n, m \in \mathbb{N}_+$ be two positive numbers and $U = (V \cup S, E)$ an $n \times m$ grid with the vertices of S connected to the vertices of the borders of the grid, such that each vertex in the corner of the grid is adjacent to two vertices in S and all other vertices on the borders are adjacent to exactly one vertex in S.*

Then $|\mathcal{R}^U_{min}| \geq n \cdot 2^{n(m-1)} + m \cdot 2^{m(n-1)} - n \cdot m$.

This means that the number of minimal recurrent configurations grows exponentially with both the height of the grid as well as the width of the grid.

Proof: As there are exactly as many minimal recurrent configurations on U as there are S-DAGs on U, we will count a subset of S-DAGs on U to get our lower bound. We will refer to vertices $v \in V$ via their coordinates in the grid, starting with $(0,0) \in V$ to $(n-1, m-1) \in V$.

For $k \in \{0, \ldots, n-1\}$ we call a directed graph $G = (V \cup S, E')$ *k-divided* if

$$\forall (i,j) \in V : i < k \wedge i+1 < n \Rightarrow ((i,j),(i+1,j)) \in E', \tag{9}$$

$$i \geq k \wedge i+1 < n \Rightarrow ((i+1,j),(i,j)) \in E' \tag{10}$$

and $\forall s \in S : outdeg_G(s) = 1$.

See Figure 4 for an example.

It is easy to see that a k-divided directed graph is always a DAG whose sources are S, i.e each k-divided graph G is in $\mathcal{D}^{U_{n,m}}_S$:

If G contained a cycle and we started going round the circle with the edge $((i,j),(i+1,j))$ we would eventually need to get back from a vertex with first component $i+1$ to a vertex with first component i, *i.e.* G would also need to contain an edge $((i+1,k),(i,k))$ which contradicts our definition.

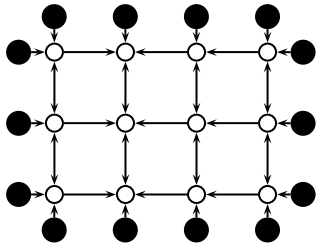


Fig. 4: A 1-divided 3×4 grid. No matter how the directions of the vertical edges between vertices $v \in V$ are chosen, the resulting graph is a DAG with sources in S.

This means that for each edge $\{(i,j),(i,j+1)\} \in E$ we can choose whether to include $((i,j),(i,j+1))$ or $((i,j+1),(i,j))$ in E', which gives us 2^{nm-1} possibilities to choose a k-divided graph for a given k.

As we have n different possibilities for k, this makes $n2^{n(m-1)}$ different S-DAGs on $U_{n,m}$.

Defining analogously l-split directed graphs for $l \in \{0,\dots,m-1\}$ gives us $m2^{m(n-1)}$ different S-DAGs.

The only graphs counted twice are graphs which are k-divided as well as l-split for some numbers k and l; these are nm different graphs, and we get $n2^{n(m-1)} + m2^{m(n-1)} - nm$ different S-DAGs on $U_{n,m}$. □

Apart from counting minimal recurrent configurations, we can use the relation between DAGs on U and minimal recurrent configurations to find recurrent configurations such that the relaxation of the sum of these configurations takes as few firings as possible.

To do so, we first take a look at DAGs with a set of sources different from S and define a DAG Game on these graphs which corresponds directly with the process of vertices firing in the Chips Firing Game on the same underlying graph..

4 The DAG Game

The DAG Game is played with directed acyclic graphs on a CFG-graph $U = (V \cup S, E)$. The simple rule is as follows:

We start with a DAG $G_1 = (V \cup S, E_1)$ on U. In the next step, we take a source v of G_1 which does not lie in S (if such a source exists) and turn it into a sink by switching the directions of all edges incident to v.

In other words the resulting graph $G_2 = (V \cup S, E_2)$ is defined by

$$\forall u,u' \in V : (u,u') \in E_2 \iff ((u,u') \in E_1 \wedge u \neq v \neq u') \vee ((u',u) \in E_1 \wedge u' = v) \tag{11}$$

See Figure 5 for an example.

If G_2 contained a cycle C this cycle could not contain the sink v; since no edges which are not incident to v have been changed C would also be a cycle in G_1 which contradicts G_1 being a DAG. Therefore G_2 is a DAG, too.

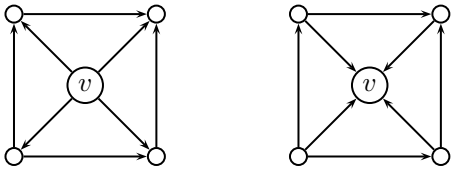


Fig. 5: The central vertex v gets turned into a sink as a step in the DAG Game. The outdegree of all adjacent vertices increases by one.

Let us look at the configurations c_{G_1} and c_{G_2}, again defined by

$$\forall v \in V : c_{G_i} = outdeg_{G_i}(v). \tag{12}$$

The only vertices for which $c_{G_2}(v) \neq c_{G_1}(v)$ is true are v and the vertices adjacent to v.

In fact, $c_{G_1}(v) = \deg(v)$ and $c_{G_2}(v) = 0$, while for each neighbor v' of v the equation $c_{G_2}(v') = c_{G_1} + 1$ is true.

This means that we get c_{G_2} by firing the vertex v in c_{G_1}.

We use the fact that we can consider the relaxation of a configuration corresponding to a DAG G with sources outside S as repeating steps of the DAG Game starting with G to show that configurations corresponding to two families of DAGs on U relax to minimal recurrent configurations.

Definition 7 *A DAG G on U whose set of sinks includes S is called a* Sup-S-DAG, *denoted $G \in \mathcal{D}^U_{S+}$.*
A DAG G on U whose set of sinks includes no vertex in S is called a Not-S-DAG, *denoted $G \in \mathcal{D}^U_{S-}$.*

We now show that configurations corresponding to Sup-S-DAGs or Not-S-DAGs always relax to minimal recurrent configurations.

Lemma 4 *Let G be a Sup-S-DAG on U.*
Then c_G relaxes to a minimal recurrent configuration.

Proof: Consider a sequence $(c_G = c_0, c_1, \ldots, c_k = (c_G)_{rel})$ such that for $0 \leq i \leq k-1$ we get $c_{i+1} = \phi_{v_i}(c_i)$ for some vertex $v_i \in V$.

We show by induction that for each $i \in \{0, \ldots k\}$ there exists a Sup-S-DAG G_i on U such that $c_i = c_{G_i}$ is true, which is obviously the case for $i = 0$.

If there exists a Sup-S-DAG G_i such that $c_i = c_{G_i}$ and there exists a vertex v_i with $\phi_{v_i}(c_i) = c_{i+1}$ this means that $outdeg_{G_i}(v_i) \geq \deg(v_i)$.

Since $outdeg_{G_i} \leq \deg(v_i)$ this means $outdeg_{G_i}(v_i) = \deg(v_i)$ and v_i is a source of G_i. Turning v_i into a sink as described above then gives us G_{i+1} such that $c_{G_{i+1}} = c_{i+1}$. All vertices in S still are sources in G_{i+1}.

The last DAG G_k has no sources outside S as c_{G_k} is stable. This means G_k is an S-DAG and $c_k = (c_G)_{rel} \in \mathcal{R}^U_{min}$.

□

Lemma 5 *Let $G = (V \cup S, E')$ be a Not-S-DAG on U.*
Then c_G relaxes to a minimal recurrent configuration.

Proof: First, we show that each vertex $v \in V$ fires at least once during the relaxation of c_G:

We define the function $p : V \to \mathbb{N}_0$ such that for all $v \in V$ $p(v)$ is the length of the longest path from a source v' of G to v, formally:

$$v \mapsto \max\{k \in \mathbb{N}_0 \mid \exists v' \in V : v' \text{ is a source in } G \text{ and there exists a path from } v' \text{ to } v \text{ in } G \text{ of length } k\}.$$

Instead of looking at the CFG dynamics for the configuration, we consider the corresponding DAG Game dynamics for G.

Assume that there exists a vertex $v \in V$ which does not turn into a sink during the relaxation of c_G and let v be a vertex with this property for which $p(v)$ is minimal.

Since for all $v' \in V$ satisfying $(v', v) \in E'$ the value $p(v')$ is less than $p(v)$ (as $p(v) = \max\{p(v') \mid (v', v) \in E'\} + 1$) this means all these vertices v' get turned into sinks during the DAG Game.

This implies that each edge (v', v) gets turned into the edge (v, v').

No vertex u' with $(v, u') \in E'$ can become a source as the edge (v, u') never gets turned to (u', v) when v cannot turn into a sink.

This means that after all vertices v' with (v', v) have been turned into sinks, there is an edge from v to each adjacent vertex u.

This means v is a source and can fire in the corresponding configuration, contradicting our assumption.

After each vertex $v \in V$ with an adjacent vertex $s \in S$ has been turned into a sink, all vertices $s \in S$ have become sources, and we get a Sup-S-DAG G' whose corresponding configuration $c_{G'}$ relaxes to a minimal recurrent configuration as shown in Lemma 4.

□

The nice thing about these lemmas is the fact that one gets a Sup-S-DAG if one switches the direction of each edge in a Not-S-DAG and vice versa, while the result of the relaxation always is a minimal recurrent configuration.

These property is quite helpful when considering the minimization of the number of firings during the addition of two recurrent configurations, as will be shown in the following section.

5 Minimizing and Maximizing Firing Vectors

In this section we will look at how often a vertex $v \in V$ can fire during the relaxation of a configuration c_G with G being a DAG on U.

We will use the result to consider the question of how many firings there will be at least when relaxing the sum of two recurrent configurations, a problem discussed by the author in [9] where we were able only to give a heuristic algorithm producing recurrent configurations whose sum causes “few” firings during the relaxation.

We can use Lemma 5 to prove a nice and, as we will see later, very helpful property of the configurations $d'_U, d_U \in \mathcal{R}^U$ defined as follows:

Definition 8 *For a CFG-graph $U = (V \cup S, E)$ we define the configuration $d'_U : V \to \mathbb{N}_0, v \mapsto \deg(v)$ and the configuration $d_U \in \mathcal{R}^U$ through$d_U = (d'_U)_{rel}$.*

As we will now deal with firing vectors, the following lemma will prove useful:

Lemma 6 *Let $c, d : V \to \mathbb{N}_0$ be configurations, not necessarily stable.*
Then $f_{c+d} = f_{c+d_{rel}} + f_d$.

Proof: Any firing sequence F for d is also a firing sequence for $c + d$, as can be easily verified.

After F we have gotten from $c + d$ to $c + d_{rel}$, so we get a firing sequence for $c + d$ by concatenating F and a firing sequence F' for $c + d_{rel}$, which means that $f_{c+d} = f_{c+d_{rel}} + d_{rel}$ is true.

□

The configuration d_U has a nice property concerning minimal recurrent configurations, as it is very easy to find two minimal recurrent configurations c and d whose relaxed sum is d_U; also, we can find a close relation between the firing vector of $c + d$ and the firing vector of the configuration $d'_U - c$, and that in fact the firing vector of $c + d$ gets minimal when the firing vector of $d'_U - c$ gets maximal.

Lemma 7 *Let $G = (V \cup S, E') \in \mathcal{D}^U_S$ be an S-DAG. Then there exists a Not-S-DAG $G' \in \mathcal{D}^U_{S-}$ such that $c_G + c_{G'} = d'_U$.*

Proof: We get $G' = (V \cup S, E'')$ by replacing each edge $(u, v) \in E'$ through the edge (v, u), formally $E'' = \{(u, v) \mid (v, u) \in E'\}$.

For all $v \in V$ each vertex u adjacent to v either satisfies $(v, u) \in E'$ or $(v, u) \in E''$, meaning that $outdeg_G(v) + outdeg_{G'}(v) = \deg(v)$.

Since all vertices $s \in S$ are sources in G, they are sinks in G', which completes the proof.

□

Theorem 3 *Let $c \in \mathcal{R}^U_{min}$ be a minimal recurrent configuration and $c' \in \mathcal{R}^U_{min}$ be the recurrent configuration satisfying $c \oplus c' = d_U$.*
Then the following is true:

(i) $c' = (d'_U - c)_{rel}$

(ii) *c' is a minimal recurrent configuration.*

(iii) $f_{c+c'} = f_{d'_U} - f_{d'_U - c}$

Proof:

(i) Let G be the firing graph for c; then $G \in \mathcal{D}^U_S$.

We turn the direction of each edge of G to get the graph G' as in Lemma 7, and get $c_G + c_{G'} = d'_U \Rightarrow c_{G'} = d'_U - c_G = d'_U - c$.

Since $G' \in \mathcal{D}^U_{S-}$ we get $(d'_U - c)_{rel} \in \mathcal{R}^U_{min}$ from Lemma 5.

Therefore $d_U = (c + (d'_U - c))_{rel} = (c + (d'_U - c)_{rel}) = c \oplus (d'_U - c)_{rel}$.

As c' is unique, we get $c' = (d'_U - c)_{rel}$.

(ii) This was shown in the proof to item (i).

(iii) This follows directly from item (i) and Lemma 6.

□

We can use Theorem 3 to find minimal recurrent configurations whose sum causes as few as possible firings and relaxes to d_U: All we have to do is maximize the number of firings during the relaxation of $d'_U - c$, which we will do presently.

To minimize the number of firings during the relaxation of the sum of two recurrent configurations one must analyze the pairs of recurrent configurations whose relaxed sum is the configuration m_U defined by $\forall v \in V : m_U(v) = \deg(v) - 1$. This means that we can find the minimal number of firings during the relaxation of the addition of two recurrent configurations if $d_U = m_U$ is true for the graph U. We will give a natural example.

Lemma 8 *For each vertex $v \in V$ and each vertex $s \in S$ let $p(v, s)$ be the length of the shortest path from v to s and $\pi(v) = \min\{p(v, s) \mid s \in S\}$ be the length of the shortest path from v to a vertex in S.*
Let G be a DAG on U and c_G be the corresponding configuration.
Then each vertex v fires at most $\pi(v)$ times during the relaxation of c_G.

Proof: Assume there is a vertex $v \in V$ which fires more than $\pi(v)$ times during the relaxaton of c_G; let v be a vertex with this property such that $\pi(v)$ is minimal.

Let $v' \in V \cup S$ be a vertex adjacent to v with $\pi(v') = \pi(v) - 1$; such a vertex exists on the shortest path from v to a vertex in S.

We consider the DAG Game corresponding to the relaxation of c_G and discuss the edge between v and v'.

As $\pi(v') < \pi(v)$ it follows that v' fires at most $\pi(v') = \pi(v) - 1$ times; this means that we have at most $\pi(v) - 1$ changes from the edge (v', v) to (v, v') during the DAG Game.

We also know that v fires at least $\pi(v) + 1$ times; this means that the edge (v, v') changes at least $\pi(v) + 1$ times to (v', v) during the DAG Game.

This means that (v, v') changes at least two times more often to (v', v) than vice versa. This is impossible, which proves the claim.

□

Lemma 9 *For each $v \in V$ $\pi(v) \in \mathbb{N}_0$ shall be defined as in Lemma 8.*
We define a sequence $(v_0, \ldots, v_{|V \cup S|-1})$ of all vertices in $V \cup S$ such that $\forall i, j \in \{0, \ldots, |V \cup S| - 1\} : i < j \Rightarrow \pi(v_i) > \pi(v_j)$.
The DAG $G = (V \cup S, E')$ defined by $(v_i, v_j) \in E' \iff \{v_i, v_j\} \in E \wedge i < j$ satisfies the following:
Each vertex $v \in V$ fires exactly $\pi(v)$ times during the relaxation of c_G.

Proof: Assume the claim is false. Let k be the smallest number such that there exists a vertex $v \in V$ satisfying $\pi(v) > k$ and v fires exactly k times during the relaxation of c_G.

Let i be the smallest number such that $\pi(v_i) > k$ and v_i fires exactly k times during the relaxation of c_G.

If $(v_i, v_j) \in E'$ we know that $\pi(v_j) \geq k$ and v_j fires at least k times during the relaxation of c_G since k is minimal.

We also know all vertices v_j with $j < i$ fire at least $k+1$ times since $\pi(v_j) \geq \pi(v_i) > k$ in these cases and i is minimal.

After all vertices v_j with $(v_j, v_i) \in E'$ have fired $k+1$ times and all vertices v_j with $(v_i, v_j) \in E'$ have fired k times and v_i has fired k times, v_i has lost $k \cdot \deg(v_i)$ chips and gained $(k+1) \cdot indeg_G(v_i) + k \cdot outdeg_G(v_i) = k \cdot \deg(v_i) + indeg_G(v_i)$ chips.

Therefore v_i contains at this moment $c_G(v_i) + indeg_G(v_i) = \deg(v_i)$ chips and can fire a $k+1$st time, which contradicts the definitions for k and v_i.

This proves the claim.

□

Note that we can get a Not-S-DAG G as described in Lemma 9 by turning the directions of all edges of the firing graph G' given by the firing sequence $(v_{|V|-1}, v_{|V|-2}, \ldots, v_0)$ which starts with vertices adjacent to vertices in S.

We now use these DAGs to minimize the number of firings during the relaxation of two recurrent configurations.

Theorem 4 *Let $m_U \in \mathcal{R}^U$ be the configuration defined by $\forall v \in V : m_U(v) = \deg(v) - 1$.*
Let $c, c' \in \mathcal{R}^U$ be two recurrent configurations. The DAGs G and G' are defined as above.

(i) Let $c'' \in \mathcal{R}^U_{min}$ be a minimal recurrent configuration satisfying $c'' \leq c'$.
Then $f_{c''+c'} \leq f_{c+c'}$.

(ii) We define $e = m_U - (c \oplus c')$. Then $c \oplus (c' \oplus e) = m$ and $f_{c+(c' \oplus e)} \leq f_{c+c'}$.

(iii) If $d_U = m_U$ then $f_{(c_G)_{rel}+c_{G'}} \leq f_{c+c'}$ is true.

Proof:

(i) This follows directly from the fact that each firing sequence for $c'' + c'$ is a firing sequence for $c + c'$ which possibly can be continued.

(ii) $f_{c+(c' \oplus e)} + f_{c'+e} = f_{c+c'+e} = f_{c+c'} + f_{(c \oplus c')+e} = f_{c+c'}$ according to Lemma 6. ($f_{(c+c')+e} = 0$ since $(c \oplus c') + e = m$ is stable.)

This proves the claim.

(iii) Items (i) and (ii) show that there exist minimal recurrent configurations $c_1, c_2 \in \mathcal{R}^U_{min}$ such that $c_1 \oplus c_2 = d_U$ and $f_{c_1+c_2} \leq f_{c+c'}$ is true.

We know $f_{c_1+c_2} = f_{d'_U} - f_{d'_U - c_1}$ from Lemma 3.

We also know there exists a DAG G'' such that $d'_U - c_1 = c_{G''}$ is true and that each vertex $v \in V$ fires at most $\pi(v)$ times during the relaxation of $c_{G''}$.

Therefore $f_{c_{G''}} \leq f_{c_G}$ and $f_{(c_G)_{rel}+c_{G'}} = f_{d'_U} - f_{c'_G} \leq f_{d'_U} - f_{c_{G''}} = f_{c_1+c_2}$ follows.

□

While there is no algorithm known which produces recurrent configurations such that the sum of these configurations produces a minimal number of firings during the relaxation for the usual Abelian Sandpile

Model, the relation between minimal recurrent configurations and S-DAGs has given us an easy way to find such configurations when the graph has a special property.

A nice example for a graph U satisfying $d_U = m_U$ is a cylindrical grid of even height with the vertices of S being above the uppermost and below the lowermost columns of the grid.

If the grid induced by V is a $n \times m$ cylindrical grid and m is even, we can compute that the relaxation of two recurrent configurations leads to at least $nm(\frac{m^2}{12} - \frac{1}{3})$ firings.

6 Results

We have shown that there exists a close relation between DAGs on U and minimal recurrent configurations (minimal with respect to the pointwise $\leq$) of the CFG played on U, which we used to get a lower bound for the number of minimal recurrent configurations of the sandpile model. Of course, this lower bound could still be improved.

We also found out that graphs corresponding to DAGs G such that either all vertices in S or no vertices in S are sources of G relax to minimal recurrent configurations, which made it easy to show that for each minimal recurrent configuration c the recurrent configuration c' such that $c \oplus c' = d_U$ is minimal recurrent itself.

We could also give a formula for the firing vector $f_{c+c'}$ and find the DAG G such that for $c = c_G$ the firing vector $f_{c+c'}$ becomes minimal. This result was used to give the minimal number of firings that occur when the sum of two recurrent configurations on a cylindrical grid gets relaxed.

These results show that the correspondence between minimal recurrent configurations and DAGs is quite helpful for analyzing recurrent configurations of Chip Firing Games. Future work could try to use this correspondence to find pairs of minimal recurrent configurations whose sum leads to as few firings as possible for underlying graphs not satisfying the condition given in Theorem 4.

Also looking at configurations c_G where G is a directed but not acyclic graph might give new insights into the structure of recurrent configurations and configurations "nearly" being recurrent.

References

[1] P. Bak, C. Tang, and K. Wiesenfeld, *Self-organized criticality: An explanation of the 1/f noise*, Phys. Rev. Lett. **59** (1987), 381–384.

[2] Y. Le Borgne and D. Rossin, *On the identity of the sandpile group*, Discrete Math. **256** (2002), no. 3, 775–790.

[3] F. Chung and R. Ellis, *A chip-firing game and dirichlet eigenvalues*, Discrete Mathematics **257** (2002), 341–355.

[4] Robert Cori and Dominique Rossin, *On the sandpile group of a graph*, European Journal of Combinatorics **21** (2000), 447–459.

[5] D. Dhar, P. Ruelle, S. Sen, and D. N. Verma, *Algebraic aspects of abelian sandpile models*, J.PHYS.A **28** (1995), 805.

[6] A. Gajardo and E. Goles, *Crossing information in two-dimensional sandpiles*, Theoretical Computer Science **369** (2006), no. 1-3, 463 – 469.

[7] L. Levine, *The sandpile group of a tree*, Eur. J. Comb. **30** (2009), no. 4, 1026–1035.

[8] S. N. Majumdar and D. Dhar, *Equivalence between the abelian sandpile model and the $q \longrightarrow 0$ limit of the potts model*, Physica A: Statistical and Theoretical Physics **185** (1992), 129–145.

[9] M. Schulz, *On the addition of recurrent configurations of the sandpile-model*, Cellular Automata (H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki, and S. Bandini, eds.), Springer Berlin/Heidelberg, 2008, pp. 236–243.

Automata 2010 — 16th Intl. Workshop on CA and DCS DMTCS proc. **AL**, 2010, 131–152

On the complexity of enumerating possible dynamics of sparsely connected Boolean network automata with simple update rules

Predrag T. Tošić

Department of Computer Science, University of Houston
501 PGH Hall, 4800 Calhoun Rd, Houston, Texas 77204-3010, USA
ptosic@uh.edu

We study how hard is to determine some fundamental properties of dynamics of certain types of network automata. We address the computational complexity of determining how many different possible dynamic evolutions can arise from some structurally very simple, deterministic and sparsely connected network automata. In this as well as our prior, related work, we try to push the limits on the underlying simplicity of two structural aspects of such network automata: (i) the uniform sparseness of their topologies, and (ii) severely restricted local behaviors of the individual agents (that is, the local update rules of the network nodes).

In this endeavor, we prove that counting the Fixed Point (FP) configurations and the predecessor and ancestor configurations in two classes of network automata, called Sequential and Synchronous Dynamical Systems (SDSs and SyDSs, respectively), are computationally intractable problems. Moreover, this intractability is shown to hold when each node in such a network is required to update according to (i) a monotone Boolean function, (ii) a symmetric Boolean function, or even (iii) a simple threshold function that is both monotone and symmetric. Furthermore, the hardness of the exact enumeration of FPs and other types of configurations of interest remains to hold even in some severely restricted cases with respect to both the network topology and the diversity (or lack thereof) of individual node's local update rules. Namely, we show that the counting problems of interest remain hard even when the nodes of an SDS or SyDS use at most two different update rules from a given restricted class, and, additionally, when the network topologies are constrained so that each node has only $c = O(1)$ neighbors for small values of constant c.

Our results also have considerable implications for other discrete dynamical system models studied in applied mathematics, physics, biology and computer science, such as Hopfield networks and spin glasses. In particular, one corollary of our results is that determining the memory capacity of sparse discrete Hopfield networks (viewed as associative memories) remains computationally intractable even when the interconnection and dependence structure among the nodes of a Hopfield network is severely restricted.

Keywords: network and graph automata, cellular automata, Hopfield networks, discrete dynamical systems, computational complexity, enumeration problems, **#P**-completeness

1 Introduction

We study certain classes of *network automata* that can be used as an abstraction of the large-scale multi-agent systems made of simple reactive agents, of ad hoc communication networks, and, more generally,

of dynamical systems whose complex dynamics stems from coupling of and interaction among their relatively simple individual components. These network or graph automata can also be viewed as a theoretical model for the computer simulation of a broad variety of computational, physical, biological and socio-technical distributed infrastructures. We are interested in the computational complexity of determining several fundamental *configuration space properties* of such network automata. The complexity of answering questions about, for instance, the existence [9], the number [67, 69] or the reachability [8] of *fixed points* (that is, the stable configurations) of an appropriate class of network automata can be argued to provide important insights into the *collective dynamics* of multi-agent systems found in distributed artificial intelligence [69], as well as other complex physical, biological, and socio-technical networks that are abstracted via those formal network automata.

In this paper, as well as in related prior work (see, e.g., [6, 7, 10, 11, 9, 66, 61, 62, 69]), the general approach has been to investigate mathematical and computational configuration space properties of such network automata, as a formal way of addressing the fundamental question: what are the possible *global behaviors* of the entire system, given the simple local behaviors of its components, and the interaction pattern among those components?

Our own focus in the context of dynamic behaviors of complex network and graph automata has been on determining *how many possible dynamics*, and in particular how many of certain types of configurations, can such discrete dynamical systems have – and *how hard* are the computational problems of determining the exact or approximate number of those various types of configurations [60, 61, 62, 63, 64, 69, 67]. We have been particularly interested in addressing the problem of counting how many *fixed point* (FP) configurations such network automata have, and how hard is the computational problem of counting those FP configurations. In this paper, we show computational intractability of determining the exact number of the fixed point configurations of sparse *Sequential and Synchronous Dynamical Systems*, as well as *discrete Hopfield networks*, whose node update rules are rather severely restricted. Moreover, we show that intractability of the exact enumeration of fixed points holds even when the maximum node degree in the underlying graph is bounded by a small constant. We also show similar intractability results for the problems of exact enumeration of all predecessors and all ancestors of a given SDS, SyDS or Hopfield network configuration. It follows from those results that, for the networked dynamical systems that can be abstracted via a class of formal network automata, a complex and generally unpredictable global dynamics can be obtained even via uniformly sparse couplings of simple, monotonic local interactions. The implication for Hopfield networks is that determining their memory capacity (when viewed as a model of associative memory) is computationally intractable, even when the structure of the underlying weight matrices of discrete, binary-valued Hopfield network is of a very particular and restricted kind.

2 Preliminaries

In this section, we define the discrete dynamical system models studied in this paper, as well as their configuration space properties. *Sequential Dynamical Systems* (SDSs) are proposed in [10, 11, 12] as an abstract model for computer simulations. These models have been successfully applied in the development of large-scale socio-technical simulations such as the *TRANSIMS* project at the Los Alamos National Laboratory [13]. A more detailed discussion of the motivation behind these models, as well as their application to large-scale simulations, can be found in [9, 61, 62] and references therein.

Definition 2.1 *A* Sequential Dynamical System *(SDS)* $\mathcal{S}$ *is a triple* (G, F, Π) *whose components are as follows:*

1. $G(V, E)$ *is a connected undirected graph without multi-edges or self-loops.* $G = G_{\mathcal{S}}$ *is referred to as the* underlying graph *of* $\mathcal{S}$*. We often use* n *to denote* $|V|$ *and* m *to denote* $|E|$*.*

2. *The state of a node* v_i*, denoted by* s_i*, takes on a value from some finite domain,* $\mathcal{D}$*. In this paper, we focus on* $\mathcal{D} = \{0, 1\}$*. We use* d_i *to denote the degree of the node* v_i*. Further, we denote by* $N(i)$ *the neighbors of node* v_i *in* G*, plus node* v_i *itself. Each node* v_i *has an associated* node update rule $f_i : \mathcal{D}^{d_i+1} \to \mathcal{D}$*, for* $1 \leq i \leq n$*. We also refer to* f_i *as the* local transition function*. The inputs to* f_i *are* s_i *and the current states of the neighbors of* v_i*. We use* $F = F_{\mathcal{S}}$ *to denote the* global map *of* $\mathcal{S}$*, obtained by appropriately composing together all the local update rules* f_i*,* $i = 1, ..., n$*.*

3. *Finally,* Π *is a permutation of the vertex set* $V = \{v_1, v_2, \ldots, v_n\}$*, specifying the order in which the nodes update their states using their local transition functions. Alternatively,* Π *can be envisioned as a total ordering on the set of nodes V. In particular, we can view the global map as a sequential composition of the local actions of each* f_i *on the respective node state* s_i*, where the node states are updated according to the order* Π*.*

The nodes are processed in the *sequential* order specified by the permutation Π. The processing associated with a node consists of computing the new value of its state according to the node's update function, and changing its state to the new value. In the sequel, we shall often slightly abuse the notation, and not explicitly distinguish between an SDS's node itself, v_i, and its state, s_i. The intended meaning will be clear from the context.

Definition 2.2 *A Synchronous Dynamical System (SyDS)* $\mathcal{S}' = (G, F)$ *is an SDS* without *the node permutation. In an SyDS, at each discrete time step, all the nodes perfectly synchronously in parallel update their state values.*

Thus, SyDSs are similar to the finite classical parallel *cellular automata* (CA) [22, 23, 25, 28, 76, 77], except that in an SyDS the nodes may be interconnected in an arbitrary fashion, whereas in a classical cellular automaton the nodes are interconnected in a regular fashion (such as, e.g., a line, a rectangular grid, or a hypercube). Another difference is that, while in the classical CA all nodes update according to the same rule, in an SyDS different nodes, in general, may use different update rules [9, 61].

Given the importance of the number of stable configurations of a Hopfield network viewed as an *associative memory* [29, 24], we next define discrete Hopfield networks. We will briefly summarize what has been known about the problem of counting their stable configurations in the subsequent sections.

Definition 2.3 *A* discrete Hopfield network *(DHN) [29] is made of* n *binary-valued nodes; the set of node states is, by convention,* $\{-1, +1\}$*. Associated to each pair of nodes* (v_i, v_j) *is (in general, real-valued)* weight, $w_{ij} \in \mathbf{R}$*. The* weight matrix *of a DHN is defined as* $W = [w_{ij}]_{i,j=1}^{n}$*. Each node also has a fixed* threshold, $h_i \in \mathbf{R}$*. A node* v_i *updates its state* x_i *from time step* t *to step* $t + 1$ *according to a* linear threshold function *of the form*

$$x_i^{t+1} \leftarrow sgn(\sum_{j=1}^{n} w_{ij} \cdot x_j^t - h_i) \tag{1}$$

where, in order to ensure that $x_i \in \{-1, +1\}$*, we define* $sgn(0) = +1$*.*

In the sequel, we will often not bother to explicitly distinguish between an S(y)DS's or DHN's node, v_i, and this node's state, denoted s_i for S(y)DSs and x_i for Hopfield networks; the meaning will be clear from the context.

In the standard DHN model, the nodes update synchronously in parallel, similarly to the classical cellular automata and the SyDSs as defined above. However, *asynchronous Hopfield networks*, where the nodes update sequentially, one at a time, have also been studied [20, 29]. In these sequential DHNs, however, it is not required that the nodes update according to a *fixed permutation* like in our SDS model. We emphasize that these differences are inconsequential insofar as the fixed points are concerned.

In most of the Hopfield networks literature, the weight matrix W is assumed *symmetric*, i.e., for all pairs of indices $\{i, j\}$, $w_{ij} = w_{ji}$ holds. A DHN is called *simple* if $w_{ii} = 0$ along the main diagonal of W for all $i = 1, ..., n$ [20].

Much of the early work on sequential and synchronous dynamical systems has primarily focused on the SDSs and SyDSs with symmetric Boolean functions as the node update rules (e.g., [6, 7, 10, 11, 67]). By *symmetric* is meant that the future state of a node does not depend on the order in which the input values of this node's neighbors are specified. Instead, the future state of v_i depends only on the states of nodes v_j in $N(i)$, i.e., on how many of v_i's neighbors are currently in the state $s_j = 1$. In particular, symmetric Boolean SyDSs correspond to totalistic (Boolean) cellular automata as defined by S. Wolfram [74, 75, 76]. The computational complexity of counting various configurations in SDSs and SyDSs with symmetric Boolean update rules is addressed in [61, 67].

We consider in this paper the SDSs, SyDSs and Hopfield networks with the local update rules that are restricted to *monotone* Boolean / binary-valued functions. Our preliminary hardness results about the counting problems in monotone Boolean SDSs and SyDSs can be found in [60, 62]. The SDSs with the local transition rules that are both monotone and symmetric are, in essence, *sequential threshold cellular automata* [65, 66, 68] that are defined over arbitrary finite graphs, as opposed to the usual regular *Cayley graphs* of the classical cellular automata [22]. We will consider the monotone update rules that are not necessarily symmetric; however, these monotone Boolean functions will be required to be of a linear threshold kind, so that our subsequent results would imply analogous results for discrete Hopfield networks [29, 30], whose update rules are, by default, always required to be linear (but not necessarily monotone) threshold functions.

We next define the notion of *monotone* Boolean functions. This definition of monotonicity readily extends to other partially ordered domains such as $\{-1, +1\}$ that has been commonly used in Hopfield networks literature.

Definition 2.4 *Given two Boolean vectors, $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$, define a binary relation "$\preceq$" as follows: $X \preceq Y$ if $x_i \leq y_i$ for all i, $1 \leq i \leq n$. An n-input Boolean function f is* monotone *if $X \preceq Y$ implies that $f(X) \leq f(Y)$.*

Notice that the notion of monotonicity given in Definition 2.4 allows us to compare only Boolean vectors of the same length.

Definition 2.5 *A configuration of a Boolean SDS $\mathcal{S} = (G, F, \Pi)$ or an SyDS $\mathcal{S}' = (G, F)$ is a vector $(b_1, b_2, \ldots, b_n) \in \{0, 1\}^n$. A configuration $\mathcal{C}$ can also be thought of as a function $\mathcal{C} : V \to \{0, 1\}^n$.*

The *global map* computed by an S(y)DS $\mathcal{S}$, denoted $F = F_{\mathcal{S}}$, specifies for each configuration $\mathcal{C}$ the next configuration reached by $\mathcal{S}$ after carrying out the updates of all the node states, whether in parallel or in the order given by Π. Thus, the map $F_{\mathcal{S}} : \{0, 1\}^n \to \{0, 1\}^n$ total function on the set of

global configurations. This function therefore defines the dynamics of $\mathcal{S}$. We say that $\mathcal{S}$ moves from a configuration $\mathcal{C}$ to a configuration $F_{\mathcal{S}}(\mathcal{C})$ in a single transition step. Alternatively, we say that S(y)DS $\mathcal{S}$ moves from a configuration $\mathcal{C}$ at time t to a configuration $F_{\mathcal{S}}(\mathcal{C})$ at time $t+1$. Assuming that each node update function f_i is computable in time polynomial in the size of the description of $\mathcal{S}$, each transition step will also take polynomial time in the size of the S(y)DS's description.

Definition 2.6 *The* configuration space *(also called* phase space*)* $\mathcal{P}_{\mathcal{S}}$ *of an SDS or SyDS* $\mathcal{S}$ *is a directed graph whose nodes are configurations and whose directed edges capture* transitions *from a configuration to its successor configuration. More formally, there is a vertex in* $\mathcal{P}_{\mathcal{S}}$ *for each global configuration of* $\mathcal{S}$*. There is a directed edge from a vertex representing configuration* $\mathcal{C}'$ *to that representing configuration* $\mathcal{C}$ *if* $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$.

Note that, since an SDS or SyDS is deterministic, each vertex in its phase space has the out-degree of 1. Since the domain $\mathcal{D}$ of state values is assumed finite, and the number of nodes in the S(y)DS is finite, the number of configurations in the phase space is also finite. If the size of the domain (that is, the number of possible states of each node) is $|\mathcal{D}|$, then the number of global configurations in $\mathcal{P}_{\mathcal{S}}$ is $|\mathcal{D}|^n$.

We next define some prominent types of configurations that are of particular interest insofar as capturing the important qualitative (and quantitative) properties of a discrete dynamical system's global behavior (that is, its dynamics).

Definition 2.7 *Given two configurations* $\mathcal{C}'$ *and* $\mathcal{C}$ *of an SDS or SyDS* $\mathcal{S}$*, configuration* $\mathcal{C}'$ *is a* PREDECESSOR *of* $\mathcal{C}$ *if* $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$*, that is, if* $\mathcal{S}$ *moves from* $\mathcal{C}'$ *to* $\mathcal{C}$ *in one global transition step. Similarly,* $\mathcal{C}'$ *is an* ANCESTOR *of* $\mathcal{C}$ *if there is a positive integer* $t \geq 1$ *such that* $F_{\mathcal{S}}{}^{t}(\mathcal{C}') = \mathcal{C}$*, that is, if* $\mathcal{S}$ *evolves from* $\mathcal{C}'$ *to* $\mathcal{C}$ *in one or more transitions.*

In particular, a predecessor of a given configuration is a special case of an ancestor.

Definition 2.8 *A configuration* $\mathcal{C}$ *of an S(y)DS* $\mathcal{S}$ *is a* GARDEN OF EDEN (GE) *configuration if* $\mathcal{C}$ *does not have a predecessor.*

Definition 2.9 *A configuration* $\mathcal{C}$ *of an S(y)DS* $\mathcal{S}$ *is a* FIXED POINT (FP) *configuration if* $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}$*, that is, if the transition out of* $\mathcal{C}$ *is back to* $\mathcal{C}$ *itself.*

Note that a fixed point is a configuration that is its own predecessor. Also note, that the fixed point configurations are also often referred to as *stable configurations* (esp. in the Hopfield networks literature); we will use the two terms interchangeably throughout the paper.

2.1 The Basics of Computational Complexity of Counting

We next define the computational complexity classes pertaining to the counting problems that we shall work with in the sequel. We also define the notion of *(weakly) parsimonious reductions* that are used to reduce one counting problem to another.

Definition 2.10 *A counting problem* Ψ *belongs to the class* **#P** *if there exists a polynomial time bounded* nondeterministic Turing machine (NTM) *such that, for each instance* I *of* Ψ*, the number of nondeterministic computational paths this NTM takes that lead to acceptance of this problem instance equals the number of solutions of* $\mathrm{I}(\Psi)$.

For an alternative but equivalent definition of the class **#P** in terms of *polynomially balanced relations*, we refer the reader to [51].

The hardest problems in the class **#P** are the **#P**-complete problems.

Definition 2.11 *A counting problem* Ψ *is* **#P***-complete if and only if (i) it belongs to the class* **#P***, and (ii) it is* hard *for that class, i.e.,* every *other problem in* **#P** *is efficiently reducible to* Ψ.

Thus, if we could solve any particular **#P**-complete problem in (deterministic) polynomial time, then all the problems in class **#P** would be solvable in (deterministic) polynomial time, and the entire class **#P** would collapse to **P**.[(i)]

As one would expect, the counting versions of the standard decision **NP**-complete problems, such as SATISFIABILITY or HAMILTON CIRCUIT, are **#P**-complete [51]. What is curious, however, is that the counting versions of some tractable decision problems, such as BIPARTITE MATCHING or MONOTONE 2CNF SATISFIABILITY, are also **#P**-complete [71, 72].

Definition 2.12 *Given two problems* Π *and* $\Pi^{'}$, *a* PARSIMONIOUS REDUCTION *from* Π *to* $\Pi^{'}$ *is a polynomial-time transformation* g *that preserves the number of solutions; that is, if an instance* I *of* Π *has* n_I *solutions, then the corresponding instance* g(I) *of* $\Pi^{'}$ *also has* $n_{g(I)} = n_I$ *solutions.*

In practice, one often resorts to reductions that are "almost parsimonious", in a sense that, while they do not exactly preserve the number of solutions, n_I in the previous definition can be efficiently recovered from $n_{g(I)}$.

Definition 2.13 *Given two problems* Π *and* $\Pi^{'}$, *a* WEAKLY PARSIMONIOUS REDUCTION *from* Π *to* $\Pi^{'}$ *is a polynomial-time transformation* g *such that, if an instance* I *of* Π *has* n_I *solutions, and the corresponding instance* g(I) *of* $\Pi^{'}$ *has* $n_{g(I)}$ *solutions, then* n_I *can be computed from* $n_{g(I)}$ *in polynomial time.*

We observe that any *parsimonious* reduction is also, trivially, *weakly parsimonious*.

All of our results on the computational complexity of counting various kinds of configurations in SDSs, SyDSs and discrete Hopfield networks will be obtained by reducing counting problems about certain types of Boolean formulae that are *known to be* **#P**-complete to the problems about S(y)DSs or Hopfield nets of a desired, appropriately restricted kind. That such reductions suffice follows from the well-known property of every problem that is *hard* for a given complexity class; for the record, we state that property in the context of the class **#P** in the Lemma below.

Lemma 2.1 [51] *Given two decision problems* Π *and* $\Pi^{'}$, *if the corresponding counting problem* $\#\Pi$ *is known to be* **#P***-hard, and if there exists a (weakly) parsimonious reduction from* Π *to* $\Pi^{'}$, *then the counting problem* $\#\Pi^{'}$ *is* **#P***-hard, as well.*

3 Related Work

Various models of *cellular* and *network automata* have been studied in a variety of contexts, from unconventional models for parallel and distributed computing (e.g., [22, 47, 68]), to complex dynamical systems [24, 25, 38], to theoretical biology [43, 44]. Beside the classical (parallel) cellular automata [22, 28] and

[(i)] Strictly speaking, since **#P** is a class of *function problems* (as opposed to the classes of *decision problems* such as **P**, **NP** and **PSPACE**), if an **#P**-complete problem turns out to be solvable in deterministic polynomial time, that would imply that $\mathbf{P}^{\#\mathbf{P}} = \mathbf{P}$.

their sequential or asynchronous variants [34, 66, 68], perhaps the most studied class of models of network or graph automata are the *Hopfield networks* [29, 30].

Computational aspects of the classical Cellular Automata have been studied in many contexts. Prior to the 1980s, most of the theoretical work dealt with infinite CA and the fundamental (un)decidability results about the global CA properties. Some examples of such properties of infinite CA are surjectivity, injectivity, and invertibility of a cellular automaton's *global map*; see, e.g., [2, 49, 52]. Systematic study of other computational aspects of CA, from topological to formal language theoretic to computational complexity theoretic, was prompted in the 1980s by the seminal work of S. Wolfram [74, 75, 76]. Among other issues, Wolfram addressed the fundamental characteristics of CA in terms of their computational expressiveness and universality. He also offered the first broadly accepted classification of all CA into four qualitatively distinct classes in terms of the structural complexity of the possible computations or, equivalently, dynamical evolutions. The state of the art pertaining to a broad variety of computational properties of CA in both theoretical and experimental domains by the end of the "golden decade" of cellular automata research (the 1980's) can be found in [28].

Since most interesting global properties of sufficiently general infinite CA have been shown to be formally undecidable, the computational complexity proper (that is, as contrasted with the computability theory) has been mainly concerned with the computational aspects of *finite* CA, or those pertaining to *finite subconfigurations* of infinite CA. Most work within that framework has focused on the fundamental *decision problems* about the possible CA computations. We include below a very short survey of some of the more important results in that context.

The first **NP**-complete problems for CA are shown by Green in [26]; these problems are of a general REACHABILITY flavor, i.e., they address the properties of the FORWARD DYNAMICS of CA. Kari studies the reverse dynamics, more specifically, the reversibility and surjectivity problems about CA [39]. Sutner also addresses the BACKWARD DYNAMICS problems, such as the problem of an arbitrary configuration's PREDECESSOR EXISTENCE, and their computational complexity in [57]. In the same paper, Sutner establishes the efficient solvability of the predecessor existence problem for an arbitrary CA with a *fixed neighborhood radius.* In [14], Durand solves the injectivity problem for arbitrary 2-D CA but restricted to the *finite subconfigurations* only; that paper contains one of the first results on **coNP**-completeness of a natural and important problem about CA. Furthermore, Durand addresses the REVERSIBILITY PROBLEM in the same, two-dimensional CA setting in [15]. Some good surveys on various directions of computational complexity-theoretic research on cellular automata can be found in [40, 47].

The SDS and SyDS models introduced in Section 2 are closely related to the *graph automata* (GA) models studied in [46, 50] and the *one-way cellular automata* studied in [54]. In fact, the general finite-domain SyDSs exactly correspond to the graph automata of Nichitiu and Remila in [50]. Barrett, Mortveit and Reidys [10, 11, 48] and Laubenbacher and Pareigis [45] investigate the mathematical properties of sequential dynamical systems. Barrett *et al.* study the computational complexity of several problems about the configuration space structure of SDSs and SyDSs. Those problems include the REACHABILITY, PREDECESSOR EXISTENCE and PERMUTATION EXISTENCE problems [7, 8]. Problems related to the existence of the GARDEN OF EDEN and the FIXED POINT configurations are studied in [9]. In particular, **NP**-completeness for the problem of FIXED POINT EXISTENCE (FPE) in various restricted classes of Boolean S(y)DSs is proven in [9]. However, the FPE problem becomes easy when all the nodes of a Boolean S(y)DS are required to update according to *monotone functions.*

The subarea of computational complexity that addresses counting or enumeration of various combinatorial structures dates back to the seminal work of L. Valiant in the late 1970s [71, 72]. Counting

problems naturally arise in many contexts, from approximate reasoning and Bayesian belief networks in AI (e.g., [55]), to network reliability and fault-tolerance [70], to theoretical biology [3], to *phase transitions* in statistical physics, which is a large body of work in itself (some representative references include [4, 35, 36, 38]).

It has been observed, however, that the progress in understanding the complexity of counting problems has been much slower than the progress related to our understanding of decision and search problems [27, 70]. Since the reductions used in proving counting problems hard have to preserve the number of solutions, rather than just whether a solution exists or not, they are in general more difficult to devise than the reductions used to establish, say, **NP**-completeness of the corresponding decision problems. For example, most standard reductions used to establish computational hardness of certain decision or search problems on graphs tend to "blow up" the underlying graph, thereby destroying the local structures that impact the number of those problems' solutions [27].

One area where this understanding of the complexity of counting has been particularly poor, is whether the general counting problems that are provably hard remain hard when various restrictions are placed on the problem instances [70]. Some of the relatively recent results in that context, such as those on the hardness of counting in *planar graphs* [32], and especially in *sparse graphs* [27, 70], have directly inspired our recent work (see [60, 61, 62, 67]), as well as the investigations summarized in this paper.

Counting problems naturally arise in the context of discrete dynamical systems, as well. Indeed, being able to efficiently solve certain counting problems is essential for the full understanding of the underlying dynamical system's qualitative behavior. The most obvious counting problem is to determine (or estimate) the number of *attractors* of the dynamical system [3]. As noted earlier, we refer to these attractors and other, not necessarily attracting *stable configurations* as to the *fixed points* (FPs); we do not address the issue of distinguishing among different types of those stable configurations (e.g., attractive vs. repulsive, etc.) in this paper.

For example, in the context of Hopfield networks, the interpretation of the FP count is that it tells us how many distinct *patterns* a given Hopfield network can *memorize* [3, 29, 30]. Computational complexity of counting FPs and other structures of interest in discrete Hopfield networks is addressed in [18, 19, 20]. We shall discuss in the next section how our results in this paper strengthen those in [18, 19] for the *symmetric* discrete Hopfield nets with integer weight matrices and *linear threshold* update rules.

4 Complexity of Counting Various Configurations of Monotone SDSs, SyDSs and Discrete Hopfield Networks

Our general approach to establishing the computational intractability of the counting problems of interest about SDSs, SyDSs and Discrete Hopfield Nets (DHNs) will be as follows. We first identify certain restricted variants of Boolean Satisfiability problem, whose counting versions (that is, determining how many satisfying truth assignment an arbitrary Boolean formula in a particular, restricted form has) are known to be intractable, that is, **#P**-complete. We then construct an S(y)DS or DHN and show that exactly enumerating a particular kind of such network automaton's configurations (e.g., its fixed points) is at least as hard as exactly enumerating the satisfying assignments of the Boolean formula. For showing intractability of counting problems, we use weakly parsimonious reductions that approximately preserve the number of solutions; for details on complexity of counting in general, and (weakly) parsimonious reductions from one counting problem to another in particular, we refer the reader to any standard reference on computational complexity, such as the book by C. Papadimitriou [51].

Monotone Boolean functions, formulae and circuits [73] have been extensively studied in many areas of computer science, from machine learning to connectionist models in AI to VLSI circuit design. Cellular and other types of network automata with the local update rules restricted to monotone Boolean functions have also been of a considerable interest (e.g., [9, 66]). The problem of counting FPs of *monotone* Boolean SDSs and SyDSs is originally addressed in [60, 62]. It is shown there that, in general, counting FPs of such S(y)DSs either exactly or approximately is computationally intractable. This intractability holds even for the graphs that are simultaneously bipartite, planar, and very sparse *on average* [60, 62, 64]. In particular:

Lemma 4.1 [62] *Counting exactly the fixed points of a monotone Boolean SDS or SyDS defined over a star graph, and such that the update rule of the central node of the star is given as a* MON 2CNF *formula of size $O(n)$, where n is the number of nodes in the star graph, is* **#P***-complete.*

Moreover, by the results of D. Roth [55], subsequently strengthened by S. Vadhan [70], the problem of *approximately* counting FPs in the setting as in Lemma 4.1 above is **NP**-hard [62].

To summarize, enumerating the fixed points of *monotone* Boolean SDSs and SyDSs defined on bipartite, planar and sparse on average underlying graphs *exactly* is **#P**-complete, and for any $\epsilon > 0$, *approximating* the number of FPs in such monotone S(y)DSs to within $2^{n^{1-\epsilon}}$ is **NP**-hard. Our next goal is to show that the hardness of the exact enumeration of FPs for monotone S(y)DSs holds even when the underlying graphs are required to be *uniformly sparse.* We will also argue that, as a consequence of our construction in the proof of Theorem 4.2 below, the problem of enumerating stable configurations of other types of discrete dynamical systems, such as the discrete Hopfield networks, is also in general computationally intractable. Moreover, that intractability holds for those discrete dynamical systems even when they are defined on very sparse underlying graphs or networks.

Given the importance of the number of stable configurations of a Hopfield network viewed as an *associative memory* (e.g., [24]), we next summarize what has been known about the problem of counting their stable configurations.

In [18], Floreen and Orponen establish the following two interesting results:[(ii)]

Theorem 4.1 *(i) The problem of determining the number of fixed point configurations of a simple discrete Hopfield network, with a symmetric weight matrix $W = [w_{ij}]$ such that all the weights w_{ij} are integers and $w_{ii} = 0$ along the main diagonal, is* **#P***-complete; and*

(ii) the problem of determining the number of predecessor configurations of a given configuration of a simple discrete Hopfield network, with a symmetric weight matrix $W = [w_{ij}]$ such that all the weights w_{ij} are from the set $\{-1, 0, +1\}$ and $w_{ii} = 0$ along the main diagonal, is **#P***-complete.*

For proving (i), Floreen and Orponen devise a Hopfield network that is quite dense, i.e., with many non-zero weights w_{ij}. This would correspond to an SDS or SyDS where there are, informally speaking, several nodes each of which having many neighbors. In contrast, our result in Lemma 4.1 allows only for a single node that has a large neighborhood; see [60, 62] for more details.

Prior to moving to our main results, for the sake of completeness, we state the following

Lemma 4.2 *Counting FPs of an arbitrary SDS or SyDS all of whose nodes use Boolean-valued linear threshold rules is* **#P***-complete.*

[(ii)] We slightly rephrase the statement of these results from the linear algebra language originally used in [18] into the discrete language we are using throughout this paper, in order to make the comparison and contrast with our own results clear.

4.1 Counting Configurations of Uniformly Sparse SDSs and SyDSs with Monotone Update Rules

The first major result of this paper pertains to the computational complexity of counting the fixed point configurations of monotone Boolean SDSs and SyDSs that are defined over *uniformly sparse* underlying graphs.

One of the original motivations behind our interest in that problem was to improve upon the complexity of counting results in [18, 19]. We shall show below that the result (i) from [18] discussed above can be considerably strengthened along several dimensions. That is, the hardness of counting FPs will be proven to still hold even when the following restrictions on the problem instances are *simultaneously* imposed:

– the underlying graphs will be required to be *uniformly sparse,* with no node degree exceeding 3;

– all linear threshold update rules will be restricted to *monotone* functions by disallowing negative weights;

– only two (positive) integer values for the weights will be allowed; and

– each S(y)DS node will choose one from only two allowed monotone linear threshold update rules.

We remark that, since each node of an SDS or SyDS in the Theorem that follows is required to have only $O(1)$ neighbors, the issue of *encoding* of the local update rules, that is discussed in detail in [62], is essentially irrelevant in the present context. In particular, even a truth table with one row for each combination of the values of a given node's neighbors is permissible [61, 62].

In the sequel, BOOL-MON-S(Y)DS will stand for a *monotone Boolean* SDS or SyDS.

Theorem 4.2 *Counting the fixed points of* BOOL-MON-S(Y)DS*s* exactly *is* **#P***-complete, even when all of the following restrictions on the structure of such an S(y)DS* simultaneously *hold:*

- *the monotone update rules are* linear threshold functions*;*
- *the S(y)DS is with memory, and such that, along the main diagonal,* $w_{ii} = 1$ *for all indices* i*,* $1 \leq i \leq n$*;*
- *at most two different positive integer weights are used by each local update rule;*
- *each node has at most three neighbors in the underlying graph of this S(y)DS;*
- *only two different monotone linear threshold rules are used by the S(y)DS's nodes.*

Proof: We first describe the construction of a BOOL-MON-SYDS from an instance of a *Boolean monotone 2CNF* (MON-2CNF) formula [21] such that no variable appears in more than three different clauses. We then outline why is this reduction from the problem of counting satisfying assignments of such a formula to the problem of counting FPs in the resulting SyDS *weakly parsimonious* [21].

Let's assume that a MON-2CNF Boolean formula is given, such that there are n variables, m clauses, each variable appears in at least one clause, and no variable appears in more than three clauses. In particular, these requirements together imply that $m = O(n)$, but we shall keep m and n as two distinct parameters for clarity.

The corresponding SyDS $\mathcal{S}$ is constructed as follows. To each variable in the formula corresponds a variable node, and to each clause, a clause node. In addition, a *cloned clause node* is introduced for each of the original m clause nodes. Thus, the underlying graph of $\mathcal{S}$ has exactly $n + 2m$ nodes. A variable node is adjacent to a clause node if and only if, in the Boolean formula, the corresponding variable appears

in the corresponding clause. Each clause node is adjacent to its clone. Finally, the cloned clause nodes form a ring among themselves.

Therefore, the underlying graph of this SyDS looks as in Figure 4.1.

In the sequel, we shall slightly abuse the notation and use x_i *both* to denote a variable in the Boolean formula, and the corresponding *variable node* in the S(y)DS or discrete Hopfield network we are constructing. Similarly, in this proof as well as throughout the rest of the paper, C_j will denote both clauses in the Boolean formulae and clause nodes in the S(y)DSs or Hopfield networks that are being constructed from those formulae. Again, the intended meaning will be clear from the context.

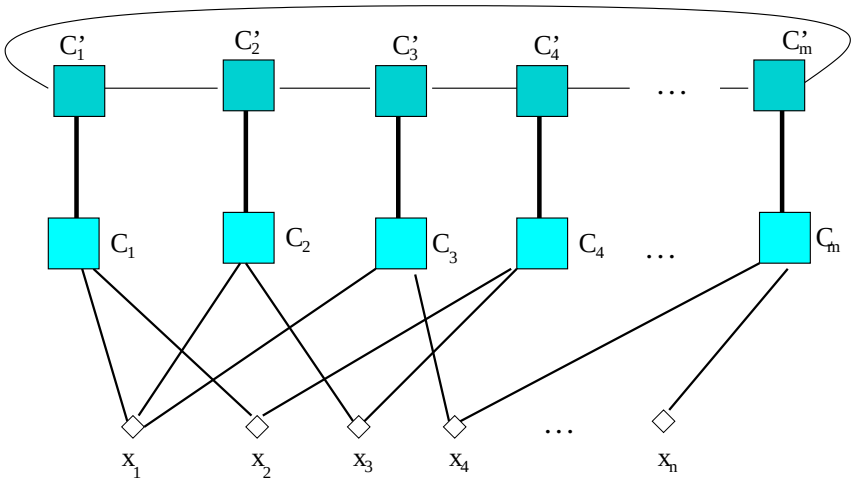


Fig. 1: Figure 4.1: *The underlying graph of a bounded-degree monotone linear threshold Boolean S(y)DS in the construction of Theorem 4.2. The original clause nodes are marked C_j, the cloned clause nodes are primed, as in C'_j, and the variable nodes are denoted by x_i.*

With these conventions in mind, we now define the update rules for the clause nodes, cloned clause nodes, and variable nodes of the constructed SyDS. The cloned clause nodes C'_j and the variable nodes x_i will update according to the Boolean *AND* rule. The original clause nodes, C_j, will update according to the following monotone linear threshold update rule:

$$C_j \leftarrow \begin{cases} 1, & \text{if } 2C'_j + C_j + x_{j_1} + x_{j_2} \geq 4 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where x_{j_1}, x_{j_2} is a shorthand for the two variable nodes that are adjacent to the clause node C_j.

The given construction can be slightly rephrased, in order to emphasize that the resulting SyDS also satisfies the *symmetry requirement* as it is usually defined in the Hopfield networks literature, namely, so

that the underlying matrix of weights is a symmetric matrix. To that end, the Boolean *AND* rule used by the cloned clause nodes can be written in an equivalent, but more "linear-threshold-like", form:

$$C'_j \leftarrow \begin{cases} 1, & \text{if } 2C_j + C'_j + C'_{j-1} + C'_{j+1} \geq 5 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Notice that the function defined in equation (3) evaluates to 1 if and only if all of its inputs are 1, and thus, indeed, the given formula is nothing but a linear-threshold-like way of writing the ordinary Boolean *AND* on four inputs. If this latter convention on how we write the update rules at the cloned clause nodes and the variable nodes is adopted, then the resulting $\mathcal{S}$ can be also viewed as a discrete Hopfield network with parallel node updates. We will turn to the related complexity of counting results in the context of Hopfield networks in the next subsection.

We now show that the reduction from the counting problem #MON-2CNF-SAT to the counting problem #FP for the constructed SyDS is, indeed, weakly parsimonious. To that end, we summarize the case analysis. If, at any time step t, one of the cloned clause nodes C'_j evaluates to 0, that will ensure that, within no more than $\frac{m}{2}$ steps, all the cloned clause nodes will become 0, and stay in state 0 thereafter. This will also cause all the original clause nodes' states C_k, and, consequently, also all the variable nodes' states x_i, to become 0, as well. Thus, if at any point a single cloned clause node's state becomes 0, the entire SyDS will eventually collapse to the "sink" fixed point 0^{n+2m}. Clearly, this sink FP does not correspond to a satisfying assignment to the original Boolean formula.

Now, the only way that no cloned clause node ever evaluates to 0 is that the following two conditions simultaneously hold:

– each C'_k and C_k is initially in the state 1, for $1 \leq k \leq m$; and

– the initial states x_i of the variable nodes are such that they correspond to a satisfying truth assignment to the variables in the original Boolean formula.

If these conditions hold, then each such global configuration $(x^n, C^m, C'^m) = (x^n_{sat}, 1^m, 1^m)$ is a fixed point of $\mathcal{S}$, where $x^n_{sat} \in \{0,1\}^n$ is a short-hand for an n-tuple of Boolean values that corresponds to a satisfying truth assignment $(x_1, ..., x_n)$ to the underlying monotone 2CNF formula. Moreover, the satisfying truth assignments of the original Boolean formula are in a one-to-one correspondence with these non-sink FPs of $\mathcal{S}$.

Since no variable in the MON-2CNF formula from which we are constructing the SyDS appears in more than three clauses, each variable node x_i in the SyDS has at most three neighbors. Since we use 2CNF, each clause node C_j has two variable node neighbors, plus one cloned clause neighbor, C'_j, for the total of three neighbors. Finally, each cloned clause node C'_j clearly has exactly three neighbors. In particular, by the result of C. Greenhill in [27], we can make the underlying graph of SyDS $\mathcal{S}$ be 3-regular, and the **#P**-completeness of the counting problem #FP will still hold.

We also observe that *only two* different monotone linear threshold functions are used in the construction above; furthermore, when the update rules are written in the form as in expressions (2) and (3), it is immediate that at most two different integer weights are used in each of those two linear threshold functions. Hence, the claim of the Theorem follows insofar as monotone linear threshold SyDSs are concerned.

Finally, by the invariance of FPs with respect to the node update ordering [48], it follows that exactly enumerating FPs of monotone linear threshold SDSs defined on uniformly sparse graphs is **#P**-complete, as well. ■

In the construction above, the SyDS dynamics from *every* starting global configurations that is not of the form $(x^n_{sat}, 1^m, 1^m)$ will eventually converge to the sink state 0^{n+2m}. In particular, the *basin of attraction* of $\mathcal{C} = 0^{n+2m}$ includes all configurations of the form $(x^n_{unsat}, 1^m, 1^m)$, where x^n_{unsat} is a shorthand for an ordered n-tuple of Boolean values that corresponds to an *unsatisfying* (i.e., falsifying) truth assignment to the corresponding variables $x_1, ..., x_n$ in the original MON-2CNF formula. The rest of the configurations in the sink's basin of attraction are such that $(C^m, C'^m) \neq (1^m, 1^m)$, where $x^n \in \{0,1\}^n$ is arbitrary.

Hence, in order to determine *exactly* the size of the basin of attraction for the sink state $\mathcal{C} = 0^{n+2m}$, that is, the number of that configuration's ancestors, we must be able to exactly determine the number of falsifying truth assignments to the original MON-2CNF Boolean formula. It is easy to see that one can find an ordering Π under which the same claim holds for the corresponding BOOL-MON-SDS. As a consequence, we have

Corollary 4.1 *The problem of counting exactly all the* ancestors *of an arbitrary configuration of a* BOOL-MON-S(Y)DS*, denoted* #ANC*, is* **#P***-hard. Moreover, this intractability result holds even when all restrictions from* Theorem 4.2 *are simultaneously imposed on the S(y)DS's structure.*

4.2 Counting Configurations of Discrete Hopfield Networks

We now turn to the corresponding hardness of counting results for discrete Hopfield networks with appropriately restricted weight matrices. We start with the problem of fixed point enumeration in the context of Hopfield nets where each of the nodes has exactly one bit of memory – namely, its own (binary-valued) current state.

Theorem 4.3 *Determining the exact number of stable configurations of a parallel or asynchronous discrete Hopfield network is* **#P***-complete even when all of the following restrictions on the weight matrix* $W = [w_{ij}]$ *simultaneously hold:*

- *the matrix is symmetric:* $w_{ij} = w_{ji}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$ *(where* $|V|$ *denotes the number of nodes in the underlying graph of this DHN);*
- $w_{ii} = 1$ *along the main diagonal for all* $i \in \{1, ..., |V|\}$*;*
- $w_{ij} \in \{0, 1, 2\}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$*;*
- *each row and each column of* W *has* at most three *(alternatively,* exactly three*) nonzero entries off the main diagonal.*

Proof sketch: In case of the DHNs whose nodes update synchronously in parallel, the claim holds by virtue of Theorem 4.2, since an SyDS that is constructed as in the proof of that theorem can also be viewed as a parallel discrete Hopfield network whose weight matrix satisfies all the above listed conditions.[(iii)] Insofar as the asynchronous DHNs whose nodes update in arbitrary sequential orders are concerned, while indeed those sequences of node updates need not be repetitions of a fixed permutation as in the corresponding SDSs, this difference can be easily shown to be immaterial insofar as the fixed point configurations are concerned. Therefore, Theorem 4.3 about discrete Hopfield networks is nothing

[(iii)] For simplicity of the argument, in this proof sketch we are ignoring the syntactic difference that the state space of a node in a Hopfield network is $\{-1, +1\}$, not $\{0, 1\}$.

but rephrasing Theorem 4.2, with parallel DHNs in place of SyDSs with monotone linear threshold update rules, and asynchronous/sequential DHNs replacing SDSs with the same kind of update rules. ■

Next, we consider the problems of enumerating predecessors as well as all ancestors of a given Hopfield network configuration. We shall establish the computational complexity of those two related counting problems in the context of *simple* DHNs, whose weight matrices satisfy $w_{ii} = 0$ for $\forall i \in \{1, ..., |V|\}$.

Before we proceed with a formal reduction from the problem #MON-2CNF-SAT to the problem #PRED of enumerating all predecessor configurations of a given DHN configuration, we establish the following additional conventions. First, the reduction will be from the MON-2CNF Boolean formulae with each variable appearing in at least one, and in at most (alternatively, exactly) four clauses. Second, we will abandon the usual convention in the Hopfield networks literature that the underlying graph is fully connected (i.e., a clique), and instead consider those pairs of vertices $\{v_i, v_j\}$ such that $w_{ij} = w_{ji} = 0$ not to be connected by an edge at all. We will require that the underlying DHN weight matrix W is *symmetric* in the usual, Hopfield network sense; as a consequence, the underlying graph of such a discrete Hopfield network will be undirected, which is also in accordance with our convention about S(y)DSs. Third, in the construction used in proving Theorem 4.2, we will eliminate the cloned clause nodes C'_j and, instead, connect the ordinary clause nodes into a ring.

We recall that, in a DHN, the set of possible states of a node is traditionally $\{-1, +1\}$ (instead of $\{0, 1\}$); while not essential, we will adopt this practice through the rest of the paper when it comes to the discrete Hopfield networks. With that in mind, we define the update rule of a clause node C_j to be

$$C_j \leftarrow \begin{cases} +1, & \text{if } 2C_{j-1} + 2C_{j+1} + x_{j_1} + x_{j_2} > 3 \\ -1, & \text{otherwise} \end{cases} \tag{4}$$

For each variable x_i in the MON-2CNF formula from which we are constructing our DHN, let a_i denote *the number of clauses in which* x_i *appears*; thus, under the stated assumptions, for every $i \in \{1, ..., |V|\}$, we have $a_i \in \{1, 2, 3, 4\}$. We now define the variable node update rules as

$$x_i \leftarrow \begin{cases} +1, & \text{if } \sum_{\{j: x_i \in C_j\}} C_j > a_i - 1 \\ -1, & \text{otherwise} \end{cases} \tag{5}$$

Thus a variable node x_i updates to $+1$ if and only if *all* of the clause nodes $C_{j(i)}$ corresponding to those clauses in the formula in which variable x_i appears are currently in the state $+1$.

Finally, we observe that the resulting weight matrix W, while symmetric and with all entries $w_{ij} \in \{0, 1, 2\}$, also has $w_{ii} = 0$ along the main diagonal; therefore, the constructed Hopfield network is *simple* (i.e., *memoryless*) [18, 20].

We are now ready to establish the third main result of this paper:

Theorem 4.4 *The problem* #PRED *of determining the exact number of predecessors of a given configuration of a* simple *discrete Hopfield network is* **#P***-complete. Moreover, this claim holds even when all of the following restrictions on the Hopfield net's weight matrix $W = [w_{ij}]$ are simultaneously imposed:*

- *the matrix is symmetric:* $w_{ij} = w_{ji}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$*;*
- $w_{ii} = 0$ *along the main diagonal for all* $i \in \{1, ..., |V|\}$*;*

- $w_{ij} \in \{0, 1, 2\}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$*;*
- *each row and each column has* at most / exactly four *nonzero entries.*

Proof sketch: The claim of the Theorem will follow from the fact that the satisfying truth assignments to the Boolean variables $x_1, ..., x_n$ in the original MON-2CNF Boolean formula are in a one-to-one correspondence with the set of all predecessors of the configuration $(+1)^{n+m}$ in the Hopfield net constructed from that formula. In the Hopfield network context, we will identify the Boolean value FALSE of a variable in the MON-2CNF formula with the corresponding DHN variable node's state -1, whereas the Boolean value TRUE of a variable in the formula will be mapped to the state $+1$ of the corresponding DHN variable node.

The case analysis is similar to that in the proof of Theorem 4.2. In particular, every configuration with at least one clause node C_j in the state (-1) will eventually converge to the sink fixed point $(x^n, C^m) = ((-1)^n, (-1)^m)$. Among the configurations of the form $(x^n, C^m) = (x^n, (+1)^m)$, those and only those such that the n-tuple x^n corresponds to a satisfying truth assignment to the original MON-2CNF Boolean formula will evolve to the other fixed point configuration, $(1^n, 1^m) = (+1)^{n+m}$. Moreover, this convergence to $(+1)^{n+m}$ is easily seen to take a single parallel transition. That is, the predecessors of $(+1)^{n+m}$ are precisely the configurations of the form $(x^n_{sat}, (+1)^m)$.

■

It immediately follows from the discussion in the proof sketch above that *all* ancestors of the configuration $\mathcal{C} = (+1)^{n+m}$ are also this configuration's predecessors; that is, the convergence from every configuration in the basin of attraction of $\mathcal{C}$ takes exactly one (global) parallel step.

Corollary 4.2 *The problem* #ANC *of determining the exact number of all ancestors of an arbitrary configuration of a simple discrete Hopfield network is, in the worst case,* **#P***-hard. Moreover, this intractability holds even when all the restrictions from* Theorem 4.4 *on the Hopfield network instances are simultaneously imposed.*

We remark that the #ANC problem for S(y)DSs and DHNs is indeed **#P**-complete whenever the *basin of attraction* of a fixed point – or, for that matter, of an arbitrary configuration that has ancestors – of the dynamical system in question is *shallow*. This shallowness, in particular, ensures that the problem of enumerating ancestors of all generations is in the class **#P**. However, for arbitrary basins of attraction that need not necessarily be shallow, the question arises, whether it can always be verified in polynomial time if one configuration is an ancestor of another configuration. In fact, the results in [8] imply that, given two arbitrary configurations $\mathcal{C}$ and $\mathcal{C}'$ of a monotone Boolean SDS or SyDS, determining whether $\mathcal{C}'$ is an ancestor of $\mathcal{C}$ (alternatively, whether $\mathcal{C}$ is *reachable* from $\mathcal{C}'$) is, in general, **PSPACE**-complete.

The implication of the results in [8] for the complexity of counting ancestors of an arbitrary configuration of a monotone Boolean S(y)DS or a discrete Hopfield network with a nonnegative and symmetric weight matrix is that the problem #ANC need not be in the class **#P**. In particular, it is an open problem whether #ANC $\in$ **#P** under the *sparseness* restrictions as in our three main results earlier in this paper. Therefore, all we can offer at this stage is a more conservative characterization of the complexity of #ANC in comparison to the complexity of #FP and #PRED – hence the **#P**-hardness, rather than **#P**-completeness, statements about the problem #ANC in Corollaries 4.1 and 4.2.

5 Summary

We have shown in [61, 62, 60, 64] that the problem of enumerating the fixed point configurations of two related classes of Boolean network automata, called Sequential and Synchronous Dynamical Systems is, in general, computationally intractable. We continue the general line of inquiry from our prior work in the present paper, as well. We now focus on those SDSs and SyDSs each of whose nodes is required to update its state according to a monotone Boolean function, and whose underlying network topologies are uniformly sparse, so that, in particular, each node has only $O(1)$ neighbors. Our main result in this paper is that exactly counting the fixed points of monotone, uniformly sparse Boolean SDSs and SyDSs such that no node has more than three neighbors is **#P**-complete. This result immediately implies similar intractability results for the sparse discrete Hopfield networks. Viewing Hopfield networks as a model of associative memory, our results imply that determining exactly how many different patterns can be stored in such an associative memory is, in general, computationally intractable. This computational intractability remains to hold even when no *inhibitive connections* (i.e., no edges with negative weights) are allowed, and, simultaneously,no row or column of the weight matrix has more than four nonzero entries. Moreover, our hardness result still holds even for those DHNs with integer weight matrices all of whose entries are from the set $0, 1, 2$.

Similarly, determining the exact size of the basin of attraction of a given stable configuration of a discrete Hopfield network with a symmetric weight matrix is equally intractable; moreover, this intractability result holds even when the Hopfield network is required to be simple, with a uniformly sparse weight matrix, and the same restrictions on the allowed values of weights w_{ij} as in our corresponding result about the enumeration of the stable (or, in the SDS terminology, fixed point) configurations.

Insofar as the future work is concerned, it needs to be pointed out that our results in this paper, as well as similar in spirit results in our prior work [60, 61, 62, 63, 67, 64, 69] all pertain to the worst-case complexity of counting the stable configurations and other structures of discrete dynamical systems. Of a considerable interest to statistical physics, connectionist AI and large-scale multi-agent systems research communities, however, is the problem of determining average complexity of the relevant decision, search and counting problems about the underlying system's dynamics.

Another important problem is that of the hardness of approximate counting of the fixed points and other types of configurations of interest; we have partially solved that problem for certain classes of underlying network topologies and node update rules [60, 62, 67, 64], but not for the particular restricted classes of topologies and update rules for which we have established the hardness of exact counting in this paper and the extended technical report [63]. Hence, the approximate counting in the settings discussed in the present paper, as far as we know, is still open.

To summarize our contribution in the present paper, the results in Theorems 4.2 - 4.4, and in particular the constructions in their corresponding proofs (see also [63] for details), clearly indicate that there are various restricted classes of uniformly sparse Boolean network automata and Hopfield networks for which exactly enumerating the stable configurations (FPs), as well as the predecessor and the arbitrary ancestor configurations, are all computationally intractable in the worst case. These hardness results have some interesting implications and interpretations – for example, in the context of pattern storage capacity of sparsely connected Hopfield networks viewed as associative memories. However, what is the average complexity of these important counting problems (and in particular, under what specific assumptions are those average or expected case problems tractable), are wide-open problems. We hope to address the average case complexity of those and other similar counting problems about discrete dynamical networks

in our future work.

Acknowledgements

Many thanks to my colleague Ricardo Vilalta, as well as Department of Computer Science and Texas Learning & Computation Center (TLC2) at University of Houston.

References

[1] M. Anthony. "Threshold Functions, Decision Lists, and the Representation of Boolean Functions", NeuroCOLT Techn. Report Series (NC-TR-96-028), January 1996

[2] S. Amoroso, Y. Patt. "Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures", *Journal of Computer and System Sciences* (JCSS), vol. 6, pp. 448 – 464, 1972

[3] R. J. Bagley, L. Glass. "Counting and Classifying Attractors in High Dimensional Dynamical Systems", *Journal of Theoretical Biology*, vol. 183, pp. 269–284, 1996

[4] F. Barahona. "On the computational complexity of Ising spin glass models", *Journal of Physics A: Mathematical and General*, vol. 15, pp. 3241 – 3253, 1982

[5] C. Barrett, B. Bush, S. Kopp, H. Mortveit and C. Reidys. "Sequential Dynamical Systems and Applications to Simulations", Technical Report, Los Alamos National Laboratory, September 1999

[6] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Dichotomy Results for Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA–UR–00–5984, 2000

[7] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Predecessor and Permutation Existence Problems for Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA–UR–01–668, 2001

[8] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Reachability problems for sequential dynamical systems with threshold functions", *Theoretical Computer Science*, vol. 295, issues 1–3, pp. 41–64, February 2003

[9] C. L. Barrett, H. B. Hunt, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns, P. T. Tosic. "Gardens of Eden and Fixed Points in Sequential Dynamical Systems", *Discrete Mathematics and Theoretical Computer Science* (DMTCS), spec. ed. Proc. AA DM–CCG, pp. 95–110, 2001

[10] C. Barrett, H. Mortveit, and C. Reidys. "Elements of a theory of simulation II: sequential dynamical systems" *Applied Mathematics and Computation*, vol. 107 (2–3), pp. 121–136, 2000

[11] C. Barrett, H. Mortveit and C. Reidys. "Elements of a theory of computer simulation III: equivalence of SDS", *Applied Mathematics and Computation*, vol. 122, pp. 325–340, 2001

[12] C. Barrett and C. Reidys. "Elements of a theory of computer simulation I: sequential CA over random graphs" *Applied Mathematics and Computation*, vol. 98, pp. 241–259, 1999

[13] R. Beckman et. al. "TRANSIMS – Release 1.0 – The Dallas-Forth Worth case study", Tech. Report LA UR 97–4502, Los Alamos National Laboratory, Los Alamos, New Mexico, 1999

[14] B. Durand. "Inversion of 2D cellular automata: some complexity results", *Theoretical Computer Science*, vol. 134 (2) , pp. 387 – 401, November 1994

[15] B. Durand. "A random **NP**-complete problem for inversion of 2D cellular automata", *Theoretical Computer Science*, vol. 148 (1) , pp. 19 – 32, August 1995

[16] B. Durand. "Global properties of 2D cellular automata", in E. Goles, S. Martinez (eds.), *"Cellular Automata and Complex Systems"*, Kluwer, Dordrecht, 1998

[17] C. Dyer. "One-way bounded cellular automata", *Information and Control*, vol. 44, pp. 54 – 69, 1980

[18] P. Floreen, P. Orponen. "On the Computational Complexity of Analyzing Hopfield Nets", *Complex Systems*, vol. 3, pp. 577–587, 1989

[19] P. Floreen, P. Orponen. "Attraction radii in binary Hopfield nets are hard to compute", *Neural Computation*, vol. 5, pp. 812–821, 1993

[20] P. Floreen, P. Orponen. "Complexity Issues in Discrete Hopfield Networks", *Neuro–COLT Technical Report Series*, NC–TR–94–009, October 1994

[21] M. R. Garey and D. S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP-completeness"*, W. H. Freeman and Co., San Francisco, California, 1979

[22] M. Garzon. *"Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks"*, Springer, 1995

[23] E. Goles, S. Martinez. *"Neural and Automata Networks: Dynamical Behavior and Applications"*, Mathematics and Its Applications series, vol. 58, Kluwer, 1990

[24] E. Goles, S. Martinez (eds.). *"Cellular Automata, Dynamical Systems and Neural Networks"*, Mathematics and Its Applications series, vol. 282, Kluwer, 1994

[25] E. Goles, S. Martinez (eds.). *"Cellular Automata and Complex Systems"*, Nonlinear Phenomena and Complex Systems series, Kluwer, 1999

[26] F. Green. "NP-Complete Problems in Cellular Automata", *Complex Systems*, vol. 1 (3), pp. 453–474, 1987

[27] C. Greenhill. "The Complexity of Counting Colourings and Independent Sets in Sparse Graphs and Hypergraphs", *Computational Complexity*, vol. 9, pp. 52–72, 2000

[28] H. Gutowitz (Editor). *"Cellular Automata: Theory and Experiment"*, North Holland, 1989

[29] J. J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities", *Proc. National Academy of Sciences* (USA), vol. 79, pp. 2554–2558, 1982

[30] J. J. Hopfield, D. W. Tank. "Neural computation of decisions in optimization problems", *Biological Cybernetics*, vol. 52, pp. 141–152, 1985

[31] B. Huberman, N. Glance. "Evolutionary games and computer simulations", *Proc. National Academy of Sciences* (USA), vol. 90, pp. 7716–7718, August 1993

[32] H. B. Hunt, M. V. Marathe, V. Radhakrishnan, R. E. Stearns. "The Complexity of Planar Counting Problems", *SIAM Journal of Computing*, vol. 27, pp. 1142–1167, 1998

[33] L.P. Hurd. "On Invertible cellular automata" *Complex Systems*, vol. 1(1), pp. 69-80, 1987

[34] T. E. Ingerson and R. L. Buvel. "Structure in asynchronous cellular automata", *Physica D: Nonlinear Phenomena*, vol. 10 (1–2), pp. 59–68, January 1984

[35] S. Istrail. "Statistical Mechanics, Three-Dimensionality and NP-completeness: I. Universality of Intracatability for the Partition Function of the Ising Model Across Non-Planar Lattices (Extended Abstract)", *Proceedings of the 32nd ACM Symposium on Theory of Computing* (STOC '00), Portland, Oregon, pp. 87 – 96, 2000

[36] M. Jerrum. "Two-dimensional monomer-dimer systems are computationally intractable", *J. Statistical Physics*, vol. 48, pp. 121–134, 1987. Erratum in vol. 59, pp. 1087–1088, 1990

[37] M. Jerrum, A. Sinclair. "Approximating the permanent", *SIAM Journal of Computing*, vol. 18, pp. 1149–1178, 1989

[38] M. Jerrum, A. Sinclair. "Polynomial-time approximation algorithms for the Ising model", *SIAM Journal of Computing*, vol. 22, pp. 1087–1116, 1993

[39] J. Kari. "Reversibility and surjectivity problems of cellular automata", *Journal of Computer and System Sciences*, vol. 48, pp. 149 – 182, 1994

[40] J. Kari. "Theory of cellular automata: A survey", *Theoretical Computer Science*, vol. 334, pp. 3 – 33, 2005

[41] R. Karp, M. Luby. "Monte-Carlo algorithms for enumeration and reliability problems", *IEEE Symposium on Foundations of Computer Science*, No. 24, pp. 56 – 64, 1983

[42] R. Karp, M. Luby. "Monte Carlo algorithms for the planar multiterminal network reliability problem", *Journal of Complexity*, vol. 1, pp. 45 – 64, 1985

[43] S. A. Kauffman. "Metabolic stability and epigenesis in randomly connected nets", *Journal of Theoretical Biology*, vol. 22, pp. 437 – 467, 1969

[44] S. A. Kauffman. "Emergent properties in random complex automata", *Physica D: Nonlinear Phenomena*, Volume 10, Issues 1–2, pp. 145–156, January 1984

[45] R. Laubenbacher and B. Pareigis. "Finite Dynamical Systems", Technical report, Department of Mathematical Sciences, New Mexico State University, Las Cruces, New Mexico, 2000

[46] B. Martin. "A Geometrical Hierarchy of Graphs via Cellular Automata", Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, August 1998

[47] M. Mitchell. "Computation in Cellular Automata: A Selected Review", in T. Gramms, S. Bornholdt, M. Gross, M. Mitchell, T. Pellizzari (editors), "*Nonstandard Computation*", pp. 95–140, Weinheim: VCH Verlagsgesellschaft, 1998

[48] H. Mortveit, C. Reidys. "Discrete sequential dynamical systems", *Discrete Mathematics*, vol. 226 (1–3), pp. 281–295, 2001

[49] J. Myhill. "The converse of Moore's Garden-of-Eden theorem", *Proc. Amer. Math. Soc.* vol. 14, pp. 685-686, 1963

[50] C. Nichitiu and E. Remila. "Simulations of Graph Automata", Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, August 1998

[51] C. Papadimitriou. "*Computational Complexity*", Addison-Wesley, Reading, Massachusetts, 1994

[52] D. Richardson. "Tessellations with local transformations", *J. of Computer and System Sciences (JCSS)*, 6, pp. 373-388, 1972

[53] C. Robinson. "*Dynamical systems: stability, symbolic dynamics and chaos*", CRC Press, New York, 1999

[54] Zs. Roka. "One-way cellular automata on Cayley graphs", *Theoretical Computer Science*, 132 (1–2), pp. 259–290, September 1994

[55] D. Roth. "On the Hardness of Approximate Reasoning", *Artificial Intelligence*, vol. 82, pp. 273–302, 1996

[56] K. Sutner. "De Bruijn graphs and linear cellular automata", *Complex Systems*, vol. 5 (1), pp. 19–30, 1990

[57] K. Sutner. "On the computational complexity of finite cellular automata", *Journal of Computer and System Sciences* (JCSS), vol. 50 (1), pp. 87–97, February 1995

[58] K. Sutner. "Computation theory of cellular automata", Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, August 1998

[59] C. Schittenkopf, G. Deco and W. Brauer. "Finite automata-models for the investigation of dynamical systems" *Information Processing Letters*, vol. 63 (3), pp. 137–141, August 1997

[60] P. Tosic. "On Counting Fixed Point Configurations in Star Networks", Advances in Parallel and Distributed Computational Models Workshop (APDCM'05), in *Proc. of the 19th IEEE Int'l Parallel & Distributed Processing Symposium*, Denver, Colorado, April 2005 (CD-Rom)

[61] P. Tosic. "On Complexity of Counting Fixed Point Configurations in Certain Classes of Graph Automata", *Electronic Colloquium on Computational Complexity*, Report ECCC–TR05–051, April 2005

[62] P. Tosic. "Counting Fixed Points and Gardens of Eden of Sequential Dynamical Systems on Planar Bipartite Graphs", *Electronic Colloquium on Computational Complexity*, Report ECCC–TR05–091, August 2005

[63] P. Tosic. "Computational Complexity of Some Enumeration Problems About Uniformly Sparse Boolean Network Automata", *Electronic Colloquium on Computational Complexity*, Report ECCC–TR06–159, 2006

[64] P. Tosic. "On the Complexity of Counting Fixed Points and Gardens of Eden in Sequential and Synchronous Dynamical Systems", *International Journal on Foundations of Computer Science* (IJFCS), vol. 17 (5), pp. 1179–1203, October 2006

[65] P. Tosic, G. Agha. "Concurrency vs. Sequential Interleavings in 1-D Threshold Cellular Automata", APDCM Workshop, in *Proc. of the 18th IEEE Int'l Parallel & Distributed Processing Symposium*, Santa Fe, New Mexico, USA, April 2004

[66] P. Tosic, G. Agha. "Characterizing Configuration Spaces of Simple Threshold Cellular Automata", *Proc. of the 6th Int'l Conference on Cellular Automata for Research and Industry* (ACRI'04), Amsterdam, The Netherlands, October 2004; Springer's *Lecture Notes in Computer Science* (LNCS) series, vol. 3305, pp. 861–870

[67] P. Tosic, G. Agha. "On Computational Complexity of Counting Fixed Points in Symmetric Boolean Graph Automata", in *Proc. of the 4th Int'l Conference on Unconventional Computation* (UC'05), Sevilla, Spain, October 2005; Springer's *Lecture Notes in Computer Science* (LNCS) series, vol. 3699, pp. 191–205

[68] P. Tosic, G. Agha. "Parallel vs. Sequential Threshold Cellular Automata: Comparison and Contrast", in *Proc. of the First European Conference on Complex Systems* (ECCS'05), paper # 251; European Complex Systems Society, Paris, France, November 2005

[69] P. Tosic, G. Agha. "On Computational Complexity of Predicting Dynamical Evolution of Large Agent Ensembles", *Proc. of the 3rd European Workshop on Multiagent Systems* (EUMAS'05), pp. 415–426, Flemish Academy of Sciences, Brussels, Belgium, December 2005

[70] S. Vadhan. "The Complexity of Counting in Sparse, Regular and Planar Graphs", *SIAM Journal of Computing*, vol. 31 (2), pp. 398–427, 2001

[71] L. Valiant. "The Complexity of Computing the Permanent", *Theoretical Computer Science*, vol. 8, pp. 189–201, 1979

[72] L. Valiant. "The Complexity of Enumeration and Reliability Problems", *SIAM Journal of Computing*, vol. 8 (3), pp. 410–421, 1979

[73] I. Wegener. *"The Complexity of Boolean Functions"*, Teubner Series in Computer Science, Wiley, 1987

[74] S. Wolfram. "Computation theory of cellular automata", *Communications in Mathematical Physics*, vol. 96, 1984

[75] S. Wolfram. "Twenty problems in the theory of cellular automata", *Physica Scripta*, T9, pp. 170-183, 1985

[76] S. Wolfram. *"Theory and applications of cellular automata"*, World Scientific, 1986

[77] S. Wolfram (ed.). *"Cellular Automata and Complexity (collected papers)"*, Addison-Wesley, 1994

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 153–172

Universality of 2-State Asynchronous Cellular Automaton with Inner-Independent Totalistic Transitions

Susumu Adachi[1] and Jia Lee[2,1] and Ferdinand Peper[1] and Hiroshi Umeo[3]

[1] *Nano ICT Group, National Institute of Information and Communications Technology, Japan*
[2] *College of Computer Science, Chong Qing University, China*
[3] *Dept. of Computer Science, Osaka Electro-Communication University, Japan*

This paper proposes a computationally universal 2-dimensional square lattice asynchronous cellular automaton, in which cells have merely two states. The transition function of a cell is a nonlinear function of the states of the living neighboring cells. This function depends on the positions of cells in the neighborhood with respect to the center cell. The neighborhood consists of cells at orthogonal or diagonal distances 1, 2, or 3 from the center cell. The proposed cellular automaton is *inner-independent* — a property according to which a cell's state does not depend on its previous state, but merely on the states of cells in its neighborhood. The asynchronous update mode used in this paper allows an update of a cell state to take place — but only so with a certain probability — whenever the cell's neighborhood states matches an element of the transition function's domain. Universality of the model is proved through the construction of three circuit primitives on the cell space, which are universal for the class of Delay-Insensitive circuits.

Keywords: asynchronous cellular automaton, inner-independent, totalistic rule

1 Introduction

Cellular Automata (CA) [18, 7, 22, 9] are dynamic systems in which the space is organized in discrete units called cells that assume one of a finite set of states. These cells are updated in discrete time steps according to a transition function, which determines the subsequent state of each cell from the state of the cells inside a certain neighborhood of the cell.

Asynchronous Cellular Automata (ACA) [10] are CA in which each cell is updated at random times. Though ACA are mostly applied to simulations of natural phenomena, there have been efforts to use them for computation, as the lack of a central clock has excellent potential for implementation by nanotechnology. The most recent among these models—and the most efficient in terms of hardware and time resources—use so-called *Delay-Insensitive (DI)* circuits that are embedded on the cell space to implement computation. DI circuits are asynchronous circuits that are robust to delays of signals [8, 20, 12, 2, 3, 16].

The number of cell states required for achieving computational universality is an important measure for the complexity of a CA model, and it is especially relevant for implementations by nanotechnology.

Researchers aim to minimize this number as much as possible, with various degrees of success: the ACA model with a traditional von Neumann neighborhood requires four cell states [14], whereas the model with Moore neighborhood in [2] and the hexagonal model in [3] both require six states. The model in [17] has cells with three states, whereby the neighborhood is von Neumann, but it requires a special type of transition function in which more than one cell needs to be updated at a time in each transition. The more recent model in [6] has cells with only two states, whereby the neighborhood is Moore, but the neighborhood radius is 2, i.e., it is defined as the 24 cells lying at orthogonal or diagonal distances 1 or 2 (Moore distance 1 or 2) from a cell.

In [5] an inner-independent totalistic rule is used in a synchronously timed 2-state CA, which also has a neighborhood defined as the 24 cells lying at orthogonal or diagonal distances 1 or 2 (Moore distance 1 or 2) from a cell. This model's transition function is totalistic and simple: if the number of state-1 cells in this neighborhood is four, the next state of a cell is 1, otherwise 0. The property of inner-independence may especially be useful for physical realizations: in [5] its relation with classical spin-glass systems is discussed.

In this paper we propose a square lattice CA with inner-independent totalistic rule, that has a larger neighborhood than the model in [5, 6], with cells at distances 1, 2, or 3 (Moore distance 1, 2, or 3). Unlike in the model in [6], however, the proposed model uses transition rules that are totalistic, making it a first in the context of asynchronous CA. Computational universality of the model is proved through formulating three primitive modules for DI circuits, and mapping them on the cell space. These modules—the so-called *P-Merge*, *Fork*, and the *R-Counter*—form a universal set for the class of DI circuits, meaning that any arbitrary DI circuit can be constructed from them. The transition function of the proposed CA model can be described in terms of 332 sub-totalistic rules in which each number is the sum of the state-1 cells in each domain depending on the distance between the cell and the center cell. In order to reduce the number of the rules, we adopt a full totalistic rule that is obtained as the linear combination of the sum of the state-1 cells in each domain. The coefficients of them are determined by a genetic algorithm. Consequently, the number of the rules is 330.

This paper is organized as follows. Section 2 describes the three primitive modules for DI circuits in more detail. The basic CA model is described in Section 3, followed by a description of the Genetic Algorithm in Section 4. Section 5 the implementation of signal exchange on the cell space, as well as of the three DI modules. This paper finishes with conclusions and a short discussion.

2 On Delay Insensitive Circuits

A DI circuit is an asynchronous circuit in which signals may be subject to arbitrary delays, without these being an obstacle to the circuit's correct operation [11]. Composed of interconnection lines and modules, a DI circuit uses signals—encoded as the change of a line's state—to transfer information from the output side of a module to the input side of another module. The speed of signals is not fixed.

A set of primitive modules from which any DI circuit can be constructed is proposed in Patra [19]. This set, consisting of the so-called *Merge*, *Fork* and *Tria*, is universal, but it suffers from the problem that the Tria requires a large number (six) of input and output lines, which is hard to implement on a CA using cells with only four neighbors each. One way around this problem is to relax some of the conditions on DI circuits, like in [21, 15], where lines are allowed to carry more than one signal at a time. The advantage of such *buffering lines* is more design freedom, and this translates into simpler structures of circuits and simpler primitive modules. This paper will employ this concept, allowing the use of primitive modules

with at most four input or output lines (Fig. 1):

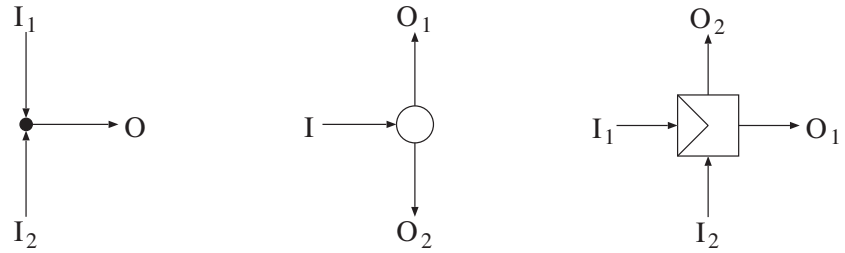


Fig. 1: Primitive modules for the DI circuits. (a) P-Merge, (b) Fork, and (c) R-Counter.

1. **P-Merge (Parallel Merge)**: A signal on input line I_1 (I_2) in Fig.1(a) is assimilated and output to O. Simultaneous signals on I_1 and I_2 are assimilated as well, and will be output as two subsequent signals to O.
2. **Fork**: A signal on input line I in Fig.1(b) is assimilated and duplicated on both output lines O_1 and O_2.
3. **R-Counter (Resettable Mod-2 Counter)**: Two subsequent signals on I_1 in Fig.1(c) are assimilated and they give rise to one output signal to O_1. This is called *Mod-2 Counter* functionality, because of the double signal required at I_1 to reinstate the initial "zero" state of the module. Alternatively, when there is one signal on each of I_1 and I_2, the module outputs a signal to O_2 after assimilating its inputs; this accounts for the *Reset* operation. A signal on only the input line I_1 keeps pending until a signal on either I_1 or I_2 is received. A signal on only the input line I_2 keeps pending until a signal on I_1 is received.

In the next section these modules will be implemented on the cell space, such that DI circuits can be constructed.

3 Rules for the model and their Construction

The ACA model consists of a 2-dimensional square array of cells, each of which can be in either of the states, 0 (dead) and 1 (alive). The neighborhood $N_{i,j}$ of a cell $C_{i,j}$ consists of the 48 cells at orthogonal or diagonal distances 1, 2 or 3 from $C_{i,j}$ (*Moore-neighborhood*).

Our previous work [6] was done on a non-totalistic 2-state ACA in which the cell neighborhood consists of 24 cells at distances 1 or 2 from the center cell (Fig. 2(a)). We have shown the computational universality of the model with inner-independent symmetric rules that are rotation-symmetric and reflection-symmetric, meaning that their equivalents rotated by multiples of 90 degrees are also transition rules, and so are their reflections.

In this paper, we use rules that are totalistic, i.e., that depend only on the number of living cells in the neighborhood, but not on the center cell's state. Such rules are called *inner-independent totalistic*. We

derive the rules in our new model from the rules in the previous model, and on the way allow for some ad-hoc rules to limit the number of rules that are generated. To this end, we first construct a so-called semi-totalistic transition rules: these are rules in which the neighborhood is divided into nine types of cells according to the scheme in Fig. 2(b). The domain of the rule table of the semi-totalistic model

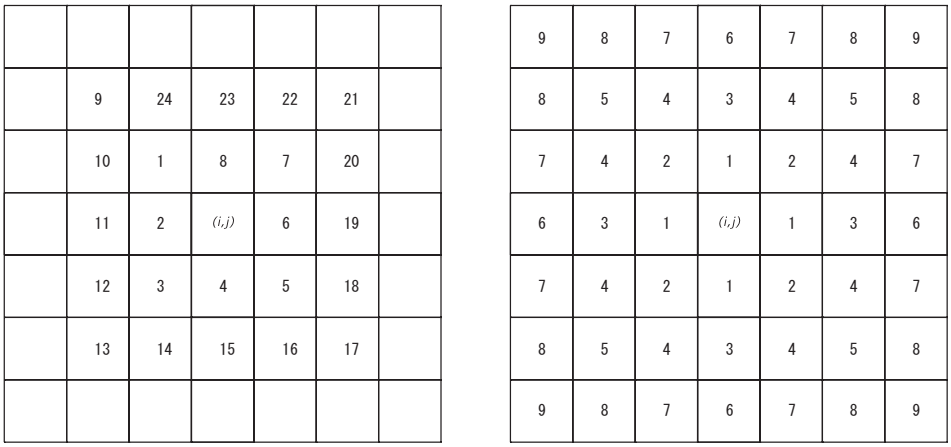


Fig. 2: (a) Moore distance 1 and 2 neighborhood of the cell (i, j) in the non-totalistic model in [6]. The numbers denote the indices of the cells in the transition rules, like in the left part of Table 1. (b) Moore distance 1, 2, and 3 neighborhood of the cell (i, j) in the semi-totalistic model in this paper. The numbers 1 to 9 denote the class to which each cell in the neighborhood belongs, like in the right part of Table 1.

consists of nine integers corresponding to the classes of the cells in a neighborhood. These nine integers indicate the number of cells of that class that are alive. The co-domain of the rule table consists of one integer that encodes the next state of the cell (0 or 1).

A non-totalistic rule is transformed into a semi-totalistic rule in roughly the following way:

- For every configuration on which a non-totalistic rule is applied, the encoding of the Left-Hand-Side of the rule as in Fig. 2(a) is written as an encoding as in Fig. 2(b). The Right-Hand-Side of the rule (0 or 1) remains the same.
- In most cases, this will give a semi-totalistic rule with a unique Left-Hand-Side (see Table 1).
- In case a conflict arises (Fig. 3), i.e., if the same Left-Hand-Side corresponds to rules with two different Right-Hand-Sides (0 and 1), then the configuration on which the rules are applied is slightly adjusted such as to change the Left-Hand-Sides of the rules and make them unique in the rule set.

After the semi-totalistic rules have been obtained, we transform them into totalistic rules. This requires that for all the possible combinations of nine integers in the rule set, we map each combination into one

Tab. 1: Transformation of non-totalistic rule of a signal into semi-totalistic rule with Moore neighborhood 1, 2, and 3. $C_{p,q}$ denotes the cells states in the Moore neighborhood. n_k denotes the number of living cells in k ($k = 1, ..., 9$) domain. C' denotes the next state.

No.	neighbor state $C_{p,q}$	C'	n_k ($k = 1...9$)	C'
1	010000000111000000000000	1	101200200	1
2	101101010000000000100000	0	321000000	0
3	000001000000001101000000	0	101200100	0
4	000001000000000101000000	0	100200100	0
5	010110000000010000000000	1	210100000	1
6	010110000000011000000000	1	211100000	1

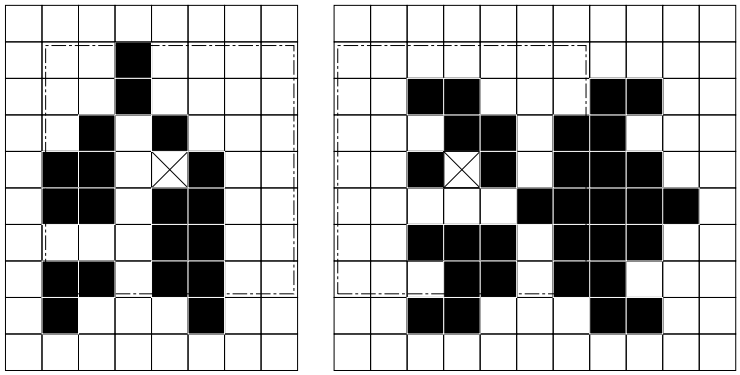


Fig. 3: An example of conflicting rules. The marked cell in the left figure must become 0, and the rule is 312402311:0. The marked cell in the right figure must become 1, and the rule is 312402311:1.

unique integer, without conflicts occurring. The transition function is expressed in totalistic form in the following way. A weighted sum X of the number of cells in each class is determined by the equation

$$X = \sum_{k=1}^{9} W_k n_k \tag{1}$$

where W_k is the weight for neighborhood cells in class k and n_k is the number of such cells in class k as given by the semi-totalistic transition rule in the rule table. Once we have determined the value of X, we associate it with an output value $F(X)$ that encodes the next state of a cell. The main issue in this transformation is of course to determine suitable weights W_k such that no conflicts in rules arise. One suitable set of values of the weights are given in table 5 in the Appendix. These values have been determined by a Genetic Algorithm, which we describe in the next section.

Updates of the cells take place asynchronously, in the way outlined in [2, 6]. According to this scheme, one cell is randomly selected from the cell space at each update step as a candidate to undergo a transition, with a probability lying between 0 and 1. If the summation of the neighbors of the selected cell match the

X-value of a transition rule, the corresponding transition is carried out.

4 Genetic Algorithm

The weights in the totalistic transition function are determined by using the so-called *parameter-free genetic algorithm (PfGA)* [23, 24, 1]. The algorithm merely uses random values or probabilities for setting almost all genetic parameters. A 'population' in the PfGA is defined as a sub-group composed of individuals, and a population size is the number of the individuals in the population. The procedure of the PfGA is as follows,

- Step 1: The first individual is generated from a whole search space randomly and is inserted into the population.
- Step 2: The second individual is generated from a whole search space randomly and is inserted into the population.
- Step 3: Two parents P_1 and P_2 are brought out from the population, and two children C_1 and C_2 are generated by multiple-point crossover operation from the parents. The number of the crossover points is determined randomly.
- Step 4: Mutation is applied to one of the children at the probability of $1/2$, in which a randomly chosen portion of the chromosome is inverted (i.e., bit-flipped).
- Step 5: By applying this selection rule, one to three selected individuals are pushed back to the population. If the population size becomes one, return back to Step 2. Otherwise, return back to Step 3.

There are four different cases in the selection rule depending on the fitness of the parents and the children as follows,

- Case 1: If both of the fitness values of the children are better than those of the parents, both of the children and the better parent are selected.
- Case 2: If both of the fitness values of the children are worse than those of the parents, the only better parent is selected as shown in case 2 of the table.
- Case 3: If the fitness of the better parent is better than that of the better child, the better parent and the better child are selected as shown in case 3 of the table.
- Case 4: If the fitness of the better child is better than that of the better parent, the only better child is selected as shown in case 4 of the table. In this case, the other individual is generated randomly and is pushed back to the population.

The population size increases in case 1, and decreases in case 2. For the most function to solve, the occurrence rate of the case 1 is less than that of the case 2. Therefore, the population size does not diverge infinitely (always less than $\sim$4). The advantages of the PfGA are compact and fast convergence due to the small population size.

The genotypes of the nine weights are encoded by nine 16-bit binary strings that are concatenated to form a string of $9 \times 16 = 144$ bits. Mutations are applied to this 144-bit string by using so-called *inverse mutations*. Suppose a string is described as $[g(1), g(2), ..., g(i-1), g(i), g(i+1), ..., g(j-1), g(j), g(j+1), ..., g(143), g(144)]$, where $g(k) = 0$ or 1, and suppose the Genetic Algorithm generates the random values $r_1 = i$ and $r_2 = j$ with $j > i$, then these bits and all bits between them are reversed, so $g(k)$ becomes $1 - g(k)$ for $k = i, ..., j$. Consequently, the mutated gene then becomes $[g(1), g(2), ..., g(i-1), 1 - g(i), 1 - g(i+1), ..., 1 - g(j-1), 1 - g(j), g(j+1), ..., g(143), g(144)]$.

The fitness function is expressed as the number of transition rules, and the algorithm aims to minimize this number. To this end, for each rule the weighted sum X in (1) is computed. This sum is the base to divide rules into two classes:

1. Normal rules: these are rules that have the same sum X and the same function value $F(X)$,
2. Forbidden rules: rules with the same sum X, but a different function value $F(X)$.

Forbidden rules are penalized by adding the value 1000 to the fitness function.

We conducted 100 trials, each consisting of 10000 generations, resulting in a number of rules of 330. The weights corresponding with these rules are:

$$\begin{aligned} W_1 &= 15012, W_2 = 11538, W_3 = 13551, \\ W_4 &= 8808, W_5 = 1804, W_6 = -4716, \\ W_7 &= -4634, W_8 = 18054, W_9 = 1612 \end{aligned}$$

However, these weights are not unique: other values that give proper evaluations of the totalistic transition function are also possible.

5 Implementing DI-Circuits on the Totalistic CA

Implementation of Signals The propagation of signals over the cell space is governed by the transition table (Table.1). Here we use the rule-based notation, rather than the totalistic transition function notation. The configuration at the left in Fig. 4 will transform through these rules via intermediate configurations into the configuration at the right in the figure, which is the same configuration as the left one, but then shifted one cell to the right.

The transition rules are designed such that they can only be applied in a strictly defined sequential order, even if the update mode is asynchronous. This ensures the reliability of signal propagation or any other operations involved in computation [4]. The design of rules also takes into account the case in which two subsequent signals appear on the same signal line (not shown here). In this case the two signals will not interfere with each other and keep a distance of at least three cells between them.

Implementation of Modules The three modules introduced in section 2 are represented on the cell space by the configurations in Fig. 5.

The configuration of the P-Merge in Fig.5(a) processes one or two input signals as shown in Fig. 6. The rule table is not shown here, but the final rule function is given by Appendix.

The configuration of the Fork in Fig.5(b) processes one input signal as shown in Fig. 7. The rule table is not shown here, but the final rule function is given by Appendix.

The configuration of the R-Counter in Fig.5(c) processes one or two signals as shown in Fig.8. The rule table is not shown here, but the final rule function is given by Appendix. Fig.8(a) illustrates the case

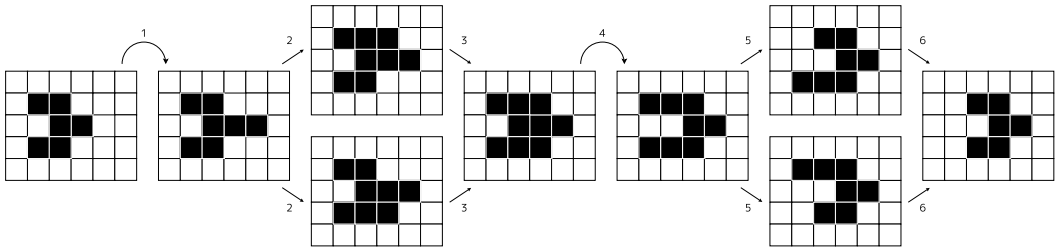
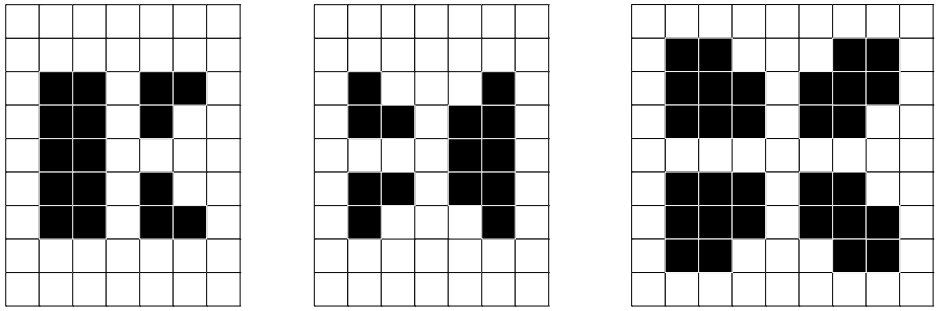


Fig. 4: The basic configuration of a signal (left figure) is transformed under the direction of transition rules 1 to 6 through the steps to the same configuration (right figure) shifted one cell to the right.



(a) (b) (c)

Fig. 5: Configurations of the primitive modules. (a) P-Merge, (b) Fork, and (c) R-Counter.

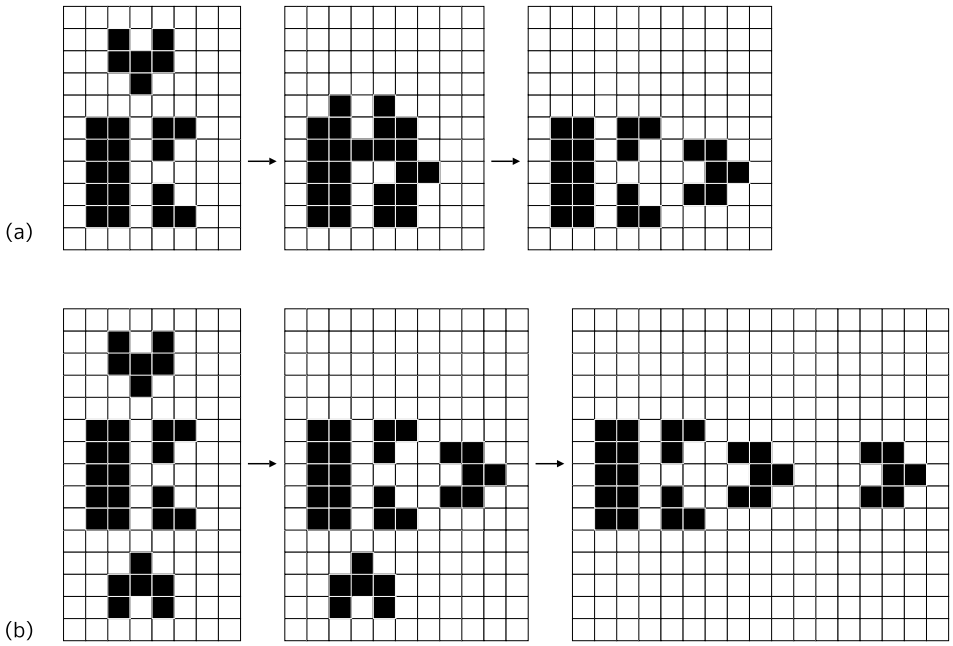


Fig. 6: (a) Processing of one single signal by the P-Merge. (b) Processing of two signals by the P-Merge.

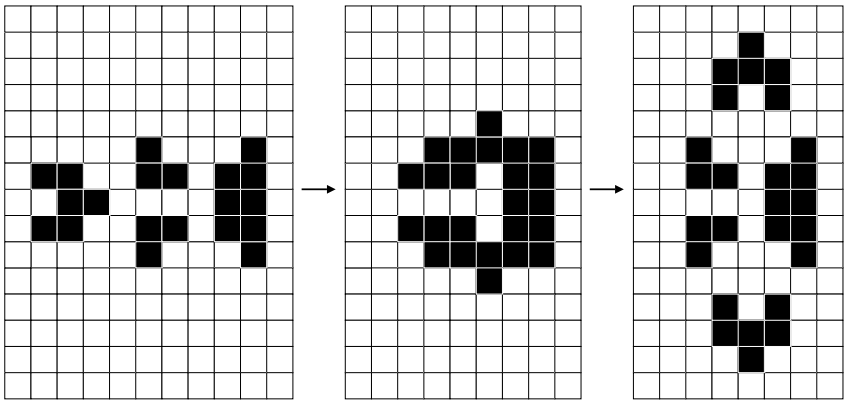


Fig. 7: Processing of a signal by the Fork, giving two output signals.

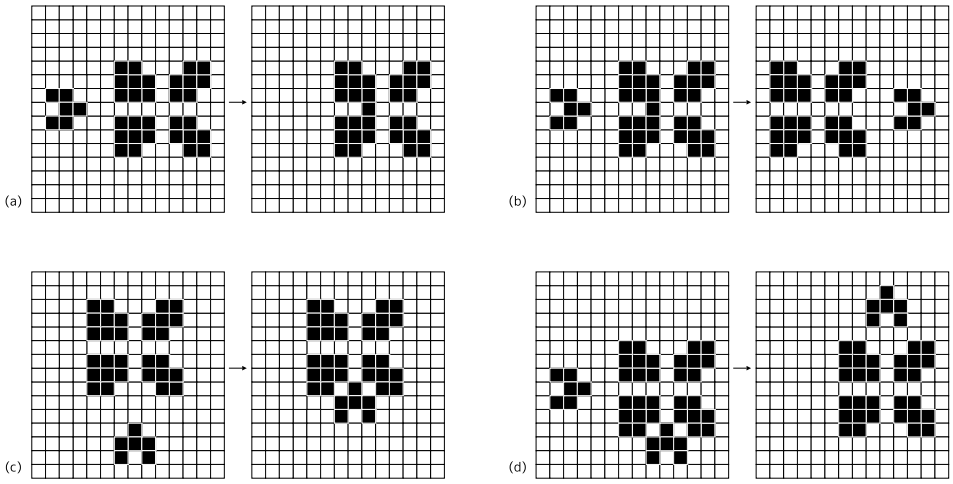


Fig. 8: Processing of signals by the R-Counter. (a) A signal input from the left line becomes a pending signal stuck inside the R-Counter. (b) Mod-2 Counter functionality: a second signal from the left input line results in a signal output to the line at the right. (c) Reset functionality: input from a reset signal to an R-Counter containing a pending signal results in (d) a signal output to the line at the top. When there are two input signals in addition to a pending signal already input from the left, the R-Counter has the choice between two possible operations. The operation illustrated here is the Mod-2 counting operation.

in which a single signal is input to the left line of the R-Counter. This signal will remain stuck in the R-Counter—we say that the signal is pending—while no further processing takes place until one more signal is received.

When one more signal is received from the same input line (Fig. 8(b)) an output signal is produced (at the right), whereby the R-Counter reverts to its initial configuration (Mod-2 Counter functionality).

When a Reset signal is input to the R-Counter in which a signal is pending (Fig. 8(c)), an output signal is produced from the line at the top, whereby the R-Counter reverts to its initial configuration (Reset functionality).

When there are signals at both input lines, while a signal input from the left is already pending, then the R-Counter has the choice to produce either of the outputs, i.e., the output at the right line or the output at the top line. One signal remains pending in both cases, but the line at which it remains pending depends on the choice made by the R-Counter. Fig. 8(d) shows the case in which the R-Counter chooses to conduct the Mod-2 counting operation, leaving the reset signal pending. The choice as to what operation is conducted by the R-Counter is arbitrary. We refer to the ability to make such a choice as *arbitration*.

In addition to the condition that the set of primitive modules is universal, it is necessary to ensure that the primitive modules can be laid out on the cell space such as to form circuits. For this it is necessary to form curves on the cell space to turn a signal left and right. Fortunately, this is an easy task, because it can be implemented by the P-Merge. One more structure that is required to form circuits is a signal crossing. Assuming that signals lack an inherent ability to cross each other, we resort to the design of a circuit specialized for this task. This circuit, shown in Fig. 9, consists of merely one R-Counter, two Forks and two P-Merges. This way of using the arbitration functionality of the R-Counter is much simpler than previously reported in literature [21], making the resulting circuit for signal crossings relatively small.

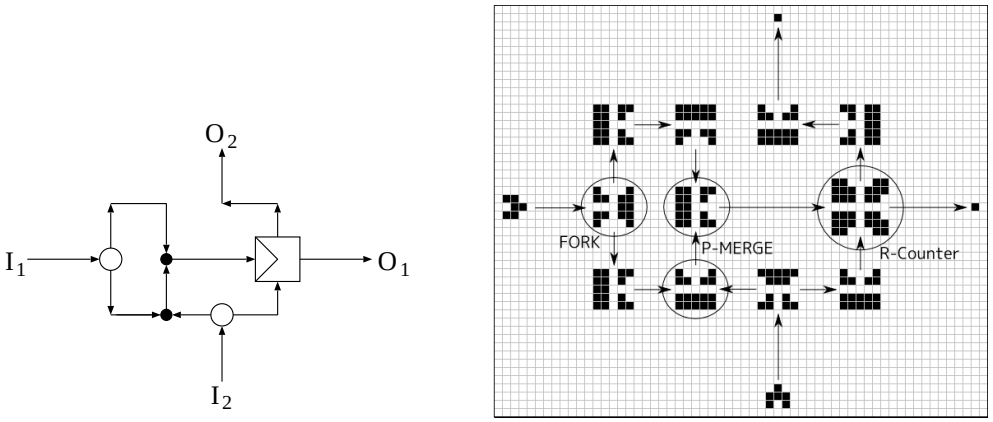


Fig. 9: (a) Circuit to cross signals without the need to intersect wires. A signal on I_1 will be directed to O_1, whereas a signal on I_2 will be directed to O_2. Simultaneous signals on I_1 and I_2 will be processed correctly, due to the arbitration functionality of the R-Counter. (b) Implementation of the crossing circuit on the ACA. The circuit is mapped on the cell space, while its topology is preserved.

The configurations of the modules thus allow us to simulate turns to the right and left of signals, as well as crossings of signals. Unlike the synchronous CA model in [5], difficult issues concerning the signal phase and the periodicity of signals on the cell space do not occur in the proposed ACA model: there is an almost unlimited freedom in laying out modules on the cell space, which is only restricted by the underlying DI circuit topology. In other words, it is quite straightforward to construct any arbitrary DI circuit on the cell space.

Next example the S-module (1-bit memory) as shown in Fig. 10 and Fig. 11.

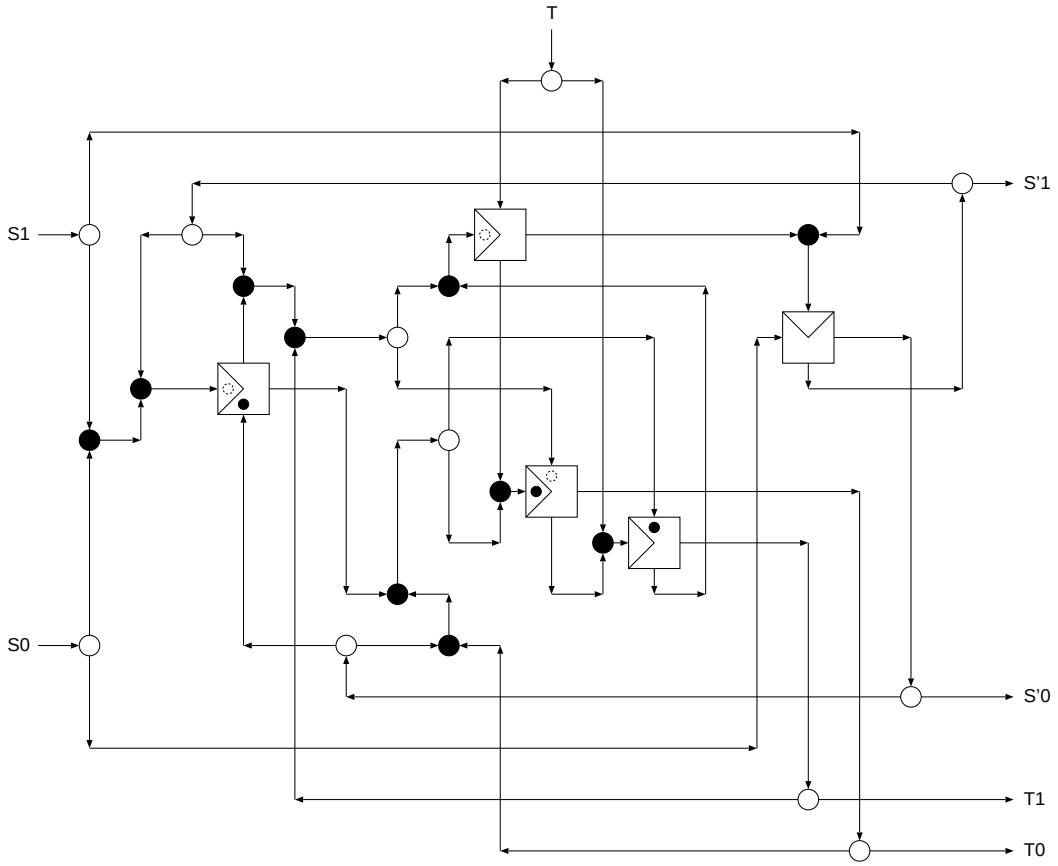


Fig. 10: Circuit of an S-module (1-bit memory).

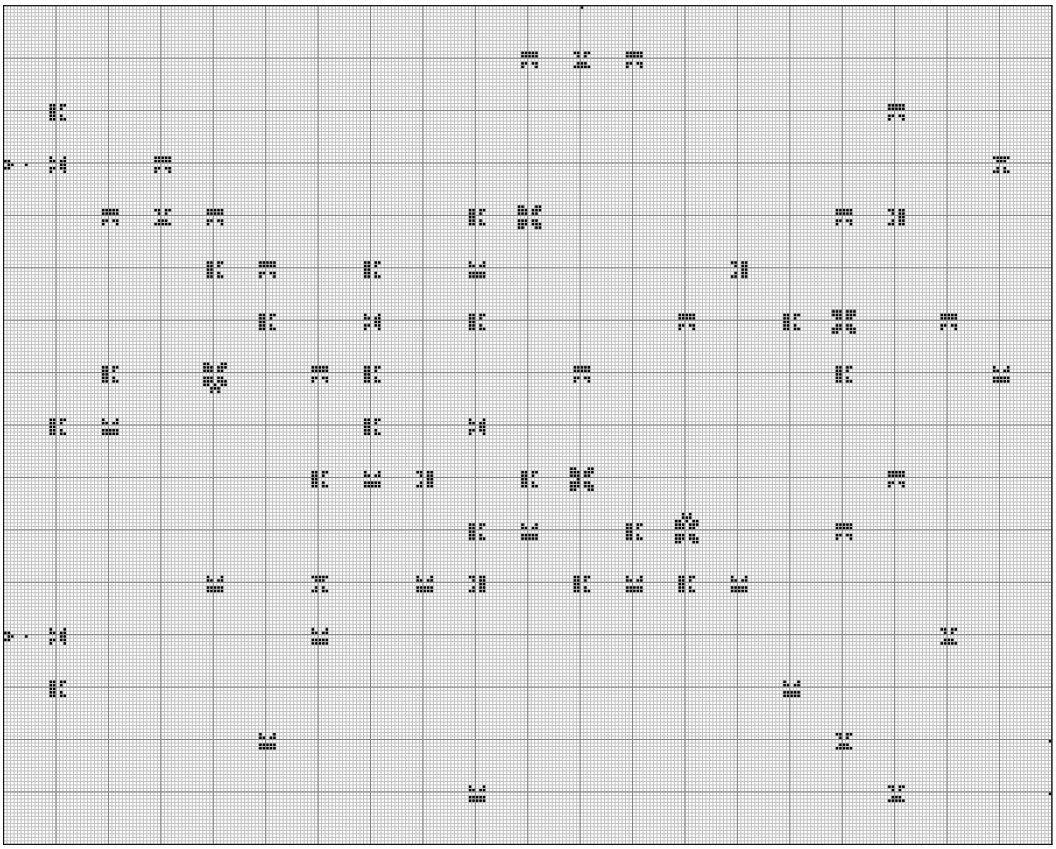


Fig. 11: Implementation of an S-module on the ACA.

6 Conclusions and Discussion

This paper proposes a 2-dimensional 2-state ACA with inner-independent totalistic rule, meaning that a cell's update does not depend on its own state but only on a linear combination of the states of the neighboring cells. The model is proved computational universal by showing how a universal set of three primitive modules can be embedded on the cell space. Since the primitive modules have at most four input or output lines each, this embedding fits well into the square lattice topology of the cell space.

The number of required transition rules in both rule table and rule function is 330, which is relatively high, when compared to other ACA models. Physical realizations of ACA models tend to require less rules, which is a strong motivation to reduce the number of rules in the model. This study, however, should be interpreted as the first proposal in which totalistic inner-independent transition rules are combined with an asynchronous mode of updating. The transition rules and cell configurations resemble those in [6]. This is a direct consequence of the totalistic rules being derived in a mostly systematic way from the non-totalistic rule. Without the ad-hoc elements included in our method, we would end up with a rule set that is many time bigger. Reducing this rule set, while being fully able to automate the method is a follow-up step that we consider. In general for such an automated approach to work, the neighborhood size employed by the resulting rule set will be larger than that of the original rule set, because of the need to avoid conflicting rules. Because of this, the number of rules will tend to increase in the rule transformation process.

References

[1] S. Adachi and H. Sawai: "Effects of migration methods in parallel distributed parameter-free genetic algorithm," Electronics and Communications in Japan (Part II: Electronics) Vol. 85 (2002) 71–80

[2] S. Adachi, F. Peper and J. Lee: "Computation by asynchronously updating cellular automata", J. Stat. Phys. 114 (1/2) (2004) 261–289

[3] S. Adachi, F. Peper and J. Lee: "Universality of Hexagonal Asynchronous Totalistic Cellular Automata", Cellular Automata, LNCS 3305 (2004) 91–100

[4] S. Adachi, J. Lee, and F. Peper: On signals in asynchronous cellular spaces. *IEICE Trans. inf. & syst.*, E87-D(3):657–668, 2004.

[5] S. Adachi, J. Lee, F. Peper and H. Umeo: "Kaleidoscope of Life: a 24-neighborhood outer-totalistic cellular automaton", Physica D 237 (2008) 800–817

[6] S. Adachi, J. Lee and F. Peper: "Universal 2-State Asynchronous Cellular Automaton with Inner-Independent Transitions", Proc. of 4th International Workshop on Natural Computing (IWNC 2009), Proceedings in Information and Communications Technology (PICT 2), Springer-Japan (2009) 107–116

[7] E. R. Berlekamp, J. H. Conway and R. K. Guy: "Wining Ways For Your Mathematical Plays", vol. 2, Academic Press, New York (1982)

[8] S. Hauck: "Asynchronous design methodologies: an overview", *Proc. IEEE*, **83** (1) (1995) 69–93

[9] A. Ilachinski: "Cellular Automata", World Scientific Publishing, Singapore (2001)

[10] T. E. Ingerson, R.L. Buvel: “Structures in asynchronous cellular automata”, Physica D 10 (1984) 59–68

[11] R. M. Keller: “Towards a theory of universal speed-independent modules”, IEEE Trans. Comput. C-23 (1) (1974) 21–33

[12] J. Lee, S. Adachi, F. Peper, K. Morita: “Embedding universal delay-insensitive circuits in asynchronous cellular spaces”, Fund. Inform. 58 (3/4) (2003) 295–320

[13] J. Lee, S. Adachi, F. Peper, K. Morita: “Asynchronous game of life”, Physica D 194 (2004) 369–384

[14] J. Lee, S. Adachi, F. Peper, and S. Mashiko. Delay-insensitive computation in asynchronous cellular automata. *Journal of Computer and System Sciences*, 70:201–220, 2005.

[15] J. Lee, F. Peper, S. Adachi, S. Mashiko: “Universal Delay-Insensitive Systems With Buffering Lines”, IEEE Trans. Circuits and Systems. 52 (4) (2005) 742–754

[16] J. Lee, F. Peper, S. Adachi, K. Morita: “An Asynchronous Cellular Automaton Implementing 2-State 2-Input 2-Output Reversed-Twin Reversible Elements”, Cellular Automata, LNCS 5191 (2008) 67–76

[17] J. Lee and F. Peper. On brownian cellular automata. In *Proc. of Automata 2008*, pages 278–291, UK, 2008. Luniver Press.

[18] J. von Neumann: “The Theory of Self-Reproducing Automata”, edited and completed by A. W. Burks, University of Illinois Press (1966)

[19] P. Patra, D. S. Fussell: “Efficient building blocks for delay insensitive circuits”, in: Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society Press, Silver Spring, MD, (1994) 196–205

[20] F. Peper, J. Lee, S. Adachi, S. Mashiko: “Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers?”, Nanotechnology 14 (4) (2003) 469–485

[21] F. Peper, J. Lee, F. Abo, T. Isokawa, S. Adachi, N. Matsui, S. Mashiko: “Fault-Tolerance in Nanocomputers: A Cellular Array Approach”, IEEE Trans. Nanotech. 3 (1) (2004) 187–201

[22] S. Wolfram: “Cellular Automata and Complexity”, Addison-Wesley, Reading, MA, USA (1994)

[23] S. Kizu, H. Sawai, and T. Endo: “Parameter-free Genetic Algorithm: GA without Setting Genetic Parameters,” Proc. of the 1997 Int. Symp. on Nonlinear Theory and its Applications 2/2 (1997) 1273–1276

[24] H. Sawai and S. Kizu: “Parameter-free Genetic Algorithm Inspired by Disparity Theory of Evolution,” Proc. of the 1997 Int. Conf. on Parallel Problem Solving from Nature (1998) 702–711

Appendix: Tables of Transition Rules

Tab. 2: Semi-totalistic rules (part 1).

No.	n_k	C'	No	n_k	C'	No	n_k	C'	No	n_k	C'
Signal											
1	101200200	1	2	221110000	1	3	222110000	1	4	341000000	0
5	102210100	0	6	101210100	0						
P-Merge											
7	101200220	1	8	101200420	1	9	101210200	0	10	101210201	0
11	101211211	0	12	101220220	1	13	101311220	0	14	101311221	0
15	101311222	0	16	101420110	1	17	101420111	1	18	101420410	1
19	102210200	0	20	102210201	0	21	102211211	0	22	102311121	0
23	102311221	0	24	112420210	0	25	112420211	0	26	112420231	0
27	113420210	0	28	121310210	0	29	121610310	1	30	122211111	0
31	122310220	1	32	122310230	1	33	141642000	1	34	203421420	0
35	203421520	0	36	212522230	1	37	221111111	1	38	221211121	1
39	222111101	1	40	222111111	1	41	222211121	1	42	222211122	1
43	222221021	1	44	223221120	1	45	223622420	1	46	223622520	1
47	231210110	1	48	232210110	1	49	232210111	1	50	232210131	1
51	232620320	0	52	242641200	0	53	313522430	1	54	321313421	0
55	321313431	0	56	321323532	0	57	322313421	0	58	322320220	0
59	322420120	0	60	322710420	1	61	323310410	1	62	331112311	1
63	331112321	1	64	331122422	1	65	331202211	1	66	332220111	1
67	332320221	1	68	332511420	1	69	333320200	1	70	333421120	1
71	333421220	1	72	333421300	1	73	334421220	0	74	334421320	0
75	341000020	0	76	341000220	0	77	341021220	0	78	341220210	0
79	341221210	0	80	341410010	0	81	341411210	0	82	341420000	0
83	341420001	0	84	343201220	0	85	343201230	0	86	424410410	1
87	433521130	0									
Fork											
88	101211221	0	89	101220221	1	90	101420231	1	91	101420400	1
92	102211221	0	93	111320222	0	94	112320222	0	95	112321310	0
96	113320221	0	97	121600200	1	98	122421331	1	99	222221020	1
100	222621130	1	101	223221020	1	102	223621130	1	103	223623440	0
104	231221121	1	105	232211220	1	106	312323331	0	107	312413321	0
108	322601420	1	109	323201400	1	110	331212311	1	111	332212432	1
112	332220100	1	113	333220100	1	114	341020220	0	115	341220200	0
116	341400222	0	117	341420110	0	118	342421220	0	119	343310220	0
120	343522230	0	121	344522230	0						

Tab. 3: Semi-totalistic rules (part 2).

No.	n_k	C'	No	n_k	C'	No	n_k	C'	No	n_k	C'
R-Counter											
122	001220221	0	123	001221242	0	124	022622400	0	125	022622401	0
126	022622410	0	127	022622510	0	128	041741484	0	129	041741584	0
130	042603620	1	131	042603630	1	132	101200222	1	133	101200422	1
134	101210211	0	135	101210221	0	136	101211110	0	137	101211311	0
138	101220442	1	139	101221242	1	140	101320221	0	141	101321221	0
142	101321231	0	143	101420442	1	144	101421440	1	145	101421441	1
146	101620641	1	147	101621640	1	148	102210211	0	149	102211110	0
150	102211311	0	151	102320221	0	152	102320231	0	153	102321231	0
154	111321321	0	155	111321332	0	156	111411321	0	157	111421331	0
158	112321321	0	159	113422210	0	160	113422310	0	161	120311221	0
162	120311222	0	163	121311221	0	164	121421640	1	165	122431321	0
166	122431322	0	167	122431331	0	168	122431332	0	169	122602620	0
170	122602630	0	171	123422130	0	172	123423200	1	173	123423201	1
174	123423310	1	175	123431321	0	176	123622410	1	177	123622510	1
178	132432331	1	179	133432331	1	180	133432332	1	181	140841484	0
182	140842484	0	183	141641284	1	184	141642284	1	185	201200422	0
186	202423200	0	187	202423201	0	188	202640642	0	189	202640742	0
190	212311110	0	191	212311210	0	192	212512220	0	193	212512241	0
194	213511210	0	195	220311221	0	196	220311222	0	197	221110111	1
198	221111221	1	199	221220121	1	200	222110111	1	201	222110121	1
202	222111221	1	203	222220121	1	204	222231131	1	205	222313421	1
206	222422220	1	207	222423220	1	208	223231131	1	209	223323230	1
210	223333130	0	211	223412220	1	212	223412230	1	213	223422120	0
214	223502321	1	215	223623210	1	216	224423400	1	217	224423401	1
218	224432120	0	219	224433120	0	220	231321221	1	221	231421231	1
222	232213321	1	223	232213331	1	224	232213431	1	225	232223432	1
226	232223532	1	227	232321221	1	228	232321322	1	229	232331231	1
230	232412421	0	231	232421221	1	232	233213321	1	233	233223432	1
234	233331221	1	235	233332231	1	236	241841284	1	237	243502521	0
238	243603420	0	239	243603430	0	240	243603521	0	241	304641642	0
242	304641742	0	243	321402411	1	244	321421231	1	245	322402411	1
246	322402421	1	247	322412422	1	248	322421231	1	249	322741562	0
250	323403420	1	251	323403430	1	252	323422210	0	253	323422220	0
254	323422221	0	255	323422241	0	256	323441242	1	257	323441342	1
258	323503521	1	259	323522241	1	260	323542452	0	261	323622400	1

Tab. 4: Semi-totalistic rules (part 3).

No.	n_k	C'	No	n_k	C'	No	n_k	C'	No	n_k	C'
R-Counter											
262	323622401	1	263	323623200	1	264	323623201	1	265	323641642	1
266	323641742	1	267	323702721	1	268	323702731	0	269	324422220	0
270	330221331	1	271	332411210	1	272	332412411	0	273	332413331	1
274	332422210	1	275	332422522	0	276	332431221	1	277	333313321	1
278	333341352	1	279	333411210	1	280	333411211	1	281	333411220	1
282	333411231	1	283	333412411	0	284	333421110	1	285	333422310	1
286	333423310	0	287	333431221	1	288	333431222	1	289	333441352	1
290	333442652	1	291	334421110	1	292	334421111	1	293	334421131	1
294	341000222	0	295	341001000	0	296	341001022	0	297	341020242	0
298	341021242	0	299	341100220	0	300	341201022	0	301	341220242	0
302	341220441	0	303	341221440	0	304	341321441	0	305	341421440	0
306	342021042	0	307	342220042	0	308	342313321	0	309	342313331	0
310	342323432	0	311	342602420	0	312	342602430	0	313	343313321	0
314	343422200	0	315	343422201	0	316	343602620	1	317	343602630	1
318	344403420	0	319	344403430	0	320	344422200	0	321	344422201	0
322	413422220	0	323	414432220	0	324	423641442	0	325	430221331	0
326	433532210	1	327	433532220	1	328	433612421	0	329	433642352	0
330	434410410	1	331	434521220	1	332	444613331	0			

Tab. 5: Totalistic transition rules.

X										$F(X)$
38715	40327	43349	49031	52266	53665	53878	56687	56900	58381	
62582	63227	67216	70238	71719	71932	72550	74162	76435	78666	
78915	80278	80527	81987	82139	82331	83120	85270	87047	88653	
89235	90052	91664	92086	94078	94926	96231	96239	96720	98712	
100023	100197	100385	100598	100763	100962	101278	102786	102890	103603	
104739	105064	105397	105596	106676	108901	113748	113936	114829	114927	
116097	116695	116910	117739	118652	122373	124936	125687	126548	128478	
130033	130382	130471	131579	131763	131916	132129	132981	134749	134803	0
134964	135187	135486	138487	138528	138998	139355	140387	140847	141432	
142990	143579	144602	145191	145845	146959	149037	149147	149358	151981	
152086	152322	153042	153593	154635	156188	156971	156999	157677	159829	
160192	160969	164826	165013	165274	165528	165532	165987	166625	167144	
168487	169636	169739	169803	169990	170035	170079	170402	171581	173024	
173101	174242	174519	174927	178564	179254	179278	181255	183842	184048	
184325	185347	186909	189635	189741	190184	191155	192135	192543	192622	
194818	197332	197590	199719	202102	202224	202733	214951	214954	225547	
227775	228502	236029	242780	247748	251569	256203	261756	265539	266472	
36911	48867	63751	66921	66975	73019	76243	76627	77263	78239	
80823	82435	83076	87579	90814	92295	94165	95219	95777	100145	
100999	101130	103129	105846	106691	107585	108593	108953	110443	111029	
111243	111783	113460	113466	113909	114441	114550	115072	116367	116981	
117979	118199	120258	120649	120830	120961	122144	123342	123900	124307	
124580	124715	125283	125347	126192	127783	127992	129604	130175	131043	
131734	131957	132818	133032	133284	134250	134381	134430	134512	136450	
136511	136893	137253	137666	138030	138312	138865	138884	140198	140525	1
141067	141186	141735	141805	142486	143008	143417	143614	144847	144946	
145508	146369	147421	147858	148197	149654	149812	151217	151774	152179	
153207	153715	153863	154316	154531	155590	158201	158349	158819	160863	
160918	161409	162293	162300	163021	163205	163363	164975	165236	165366	
165452	165475	166251	166271	167847	171069	171132	171512	173882	174201	
175844	176246	179463	181267	181398	181470	183010	183561	184121	184683	
185420	185901	186295	195021	199129	201615	202175	207115	208035	211541	
211749	212669	218638	219009	219118	228819	237627	266959	271675	304303	

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 173–186

Asymptotic behaviour of self-averaging continuous cellular automata

Heather Betel[1] and Paola Flocchini[1] and Ahmed Karmouch[1]

[1] *School of Information Technology and Engineering,*
University of Ottawa, Ottawa, Ontario, K1N 6N5, Canada.
email:{hbetel,flocchin,karmouch}@site.uottawa.ca

In this paper we continue the study of the asymptotic dynamics of fuzzy cellular automata (FCA) concentrating on a class of FCA rules called *self-averaging* rules. We begin with the elementary case. We know that all self-averaging rules converge to $\frac{1}{2}$ starting from configurations with values in $(0, 1)$; we now start the study of how fuzziness propagate in a binary CA by showing that indeed the presence of a single fuzzy value is sufficient to force some rules of this class to converge to $\frac{1}{2}$. We also study the way in which self-averaging rules converge, and we show that their fluctuations around $\frac{1}{2}$ obey, in a specific sense, a Boolean rule. We then turn our attention to the more general case of larger neighbourhoods and higher dimensions. We show the same tendency to $\frac{1}{2}$ and describe propagation of fuzziness in a class of rules. We also describe their asymptotic behaviour using a generalisation of the elementary results.

Keywords: fuzzy cellular automata, asymptotic behaviour, convergence, continuous cellular automata.

1 Introduction

Discrete dynamical systems known as cellular automata (CA) were first introduced by Von Neumann as models of self-organizing/reproducing behaviours (21). Since then, they have come to be used in fields as divergent as ecology and theoretical computer science(e.g., see (5; 14; 22)). CA are discrete in space, time and state. Kaneko introduced a modification, continuous cellular automata (or coupled map lattices), which were discrete in space and time, but continuous in state. They were conceived of as simple models exhibiting spatio-temporal chaos, and now have applications in many different areas including fluid dynamics, biology, chemistry, etc. (e.g., (12; 13)).

Introduced in (6; 7) to study the impact that state-discretization has on the behaviour of these systems, fuzzy cellular automata (FCA) are a particular type of continuous cellular automata where the local transition rule is the "fuzzification" of the local rule of a corresponding Boolean cellular automaton in disjunctive normal form(i). They have since gained currency as a modeling tool in pattern recognition (e.g., see (15; 16; 17)), and to mimick nature (e.g. (8; 20)), and have been used to investigate the effect

(i) These are not to be confused with a variant of cellular automata, also called fuzzy cellular automata, where the fuzziness refers to the local rule (e.g., see (1))

of perturbation (e.g. noisy source, computational error, mutation, etc.) on the evolution of Boolean CA (11). The asymptotic dynamics of elementary FCA (i.e., with dimension and radius one) has been observed through simulations in (9) where an empirical classification has been proposed. The asymptotic behaviour of some FCA rules has only recently been analytically studied (e.g., see (4; 10; 18; 19)); in particular it has been shown in (18; 19) that none has a chaotic dynamics, thus supporting the empirical evidence of (9). Finally, methods for controlling the dynamics of fuzzy rule 90 have been investigated in (23).

In order to study the behaviour of FCA and its relationship to the behaviour of the corresponding Boolean rules, particular classes of elementary Fuzzy CA with common properties have been identified (for example, *weighted average* rules, *self-averaging* rules, *generalized majority* rules) (4; 3). Of particular interest are the results concerning self-averaging rules whose analytical form, in the elementary case, is as follows: $f(x, y, z) = f'(y, z)x + (1 - f'(y, z))(1 - x)$ (analogously for variables y and z) for some function f'. We know that self-averaging rules with initial values in $(0, 1)$ converge to $\frac{1}{2}$. Moreover, it has been shown in (3) that when the variable averaged is y they correspond to Boolean additive rules, and they are the only rules displaying a peculiar self-oscillating behaviour around the fixed point (earlier observed in rule 90 (10)): their dynamics around their convergence point of $\frac{1}{2}$ obey the rule table of the corresponding Boolean rule.

In this paper we continue the study of this class of rules in infinite CA with any dimension and neighborhood.

We first focus on elementary self-averaging rules and we study their asymptotic behaviour when the initial configuration contains some Boolean and some fuzzy values. We know that when the initial configuration is fully Boolean they all display complex dynamics; when it is fully fuzzy (i.e., values are in the open interval $(0, 1)$) they all converge to $\frac{1}{2}$. We start the study of how fuzziness propagate in a binary CA by showing that indeed the presence of a single fuzzy value is sufficient to force some rules of this class to converge to $\frac{1}{2}$. We also conclude the study on the way in which self-averaging rules converge showing that each behaves in the proximity of $\frac{1}{2}$ following a simple CA rule and observing that such behaviour can also be observed in the corresponding Boolean rule, hidden in apparent complexity.

We then generalize some of the results on convergence and oscillation for larger neighbourhoods and higher dimensions. We show that fully fuzzy configurations will converge to $\frac{1}{2}$ and give sufficient conditions for a single fuzzy value causing an entire system to converge to $\frac{1}{2}$. Although it is difficult to provide an exhaustive description of the asymptotic fluctuations for this infinite class of rules, we do provide generalizations of the results obtained for the elementary rules and a framework for understanding what happens in the arbitrary case.

2 Definitions

A d-dimensional infinite Boolean cellular automata can be described by a quadruple $C\langle \mathbb{Z}^d, \{0, 1\}, N, g\rangle$ where: $\mathbb{Z}^d$ represents the set of *cells*; $\{0, 1\}$ is the set of possible Boolean *states* of the cells; N is the *neighbourhood* of a cell and can be defined in different ways but usually contains the cell itself plus the neighbouring cells up to a certain radius; and $g : \{0, 1\}^{|N|} \to \{0, 1\}$ is the *local function*, also called the *rule* of the automaton. Given an initial configuration, C^0, that is, a mapping $C^0 : \mathbb{Z}^d \to \{0, 1\}$, cell states are synchronously updated at each time step by the local function applied to their neighbourhoods. A configuration is the resulting map $C^t : \mathbb{Z}^d \to \{0, 1\}$ at any time t. A *finite* d-dimensional Boolean cellular automaton has a finite number of non-zero states in an infinite quiescent background. So $C^t(z) = 0$ for

all but finitely many $z \in \mathbb{Z}^d$. *Circular* cellular automata can be thought of as infinite CA with a periodic repeating pattern, or as a finite circular d-dimensional grid.

The local rule g of a Boolean CA is typically given in tabular form by listing the 2^{2q+1} binary tuples corresponding to the 2^{2q+1} possible local configurations a cell can detect in its direct neighbourhood, and mapping each tuple to a Boolean value r_i ($0 \leq i \leq 2^{2q+1}-1$): $(00\cdots00, 00\cdots01, \ldots\ldots, 11\cdots10, 11\cdots11) \rightarrow (r_0, \cdots, r_{2^{2q+1}})$. The binary representation $(r_0, \cdots, r_{2^{2q+1}})$ is often converted into the decimal representation $\sum_i r_i$, and this value is typically used as the "name" of the rule (or rule number). Let us denote by d_i the tuple mapping to r_i, and by $\mathcal{T}_1$ the set of tuples mapping to one. The local rule can also be canonically expressed in *disjunctive normal form* (DNF) as follows:

$$g(v_{-q}, \cdots, v_q) = \bigvee_{i<2^{2q+1}} r_i \bigwedge_{j=-q:q} v_j^{d_{i,(j+q)}}$$

where d_{ij} is the j-th digit, from left to right of d_i (counting from zero) and v_j^0 (resp. v_j^1) stands for $\neg v_j$ (resp. v_j) i.e. $\bigwedge_{j=-q:q} v_j^{d_{i,(j+q)}}$ will be equal to one precisely when $v_{-q}\cdots v_q$ viewed as a single binary number is equal to d_i. Cellular automata with dimension and radius one are called *elementary*.

A *fuzzy cellular automaton* (FCA) is a particular continuous cellular automaton where the local rule is obtained by *DNF-fuzzification* of the local rule of a classical Boolean CA. The fuzzification consists of a fuzzy extension of the Boolean operators AND, OR, and NOT in the DNF expression of the Boolean rule. Depending on which fuzzy operators are used, different types of fuzzy cellular automata can be defined. Among the various possible choices, we use the following: $(a \vee b)$ is replaced by $max\{1,(a+b)\}$[(ii)], $(a \wedge b)$ by (ab), and $(\neg a)$ by $(1-a)$. Whenever we talk about fuzzification, we are referring to the *DNF-fuzzification* defined above. The resulting local rule $f : [0,1]^{2q+1} \rightarrow [0,1]$ becomes a real function that generalizes the canonical representation of the corresponding Boolean CA:

$$f(v_{-q}, \cdots, v_q) = \sum_{i<2^{2q+1}} r_i \prod_{j=-q:q} l(v_j, d_{i,j+q}) \tag{1}$$

where $l(a,0) = 1-a$ and $l(a,1) = a$.

Consider, for example, elementary CA 18 whose local transition rule in tabular form is given by: $(000, 001, 010, 011, 100, 101, 110, 111) \rightarrow (0,1,0,0,1,0,0,0)$. The local transition rule can also be written in DNF form as: $g(v_{-1}, v_0, v_1) = (\neg v_{-1} \wedge \neg v_0 \wedge v_1) \vee (v_{-1} \wedge \neg v_0 \wedge \neg v_1)$, and the corresponding fuzzification is: $f(v_{-1}, v_0, v_1) = (1-v_{-1})(1-v_0)v_1 + v_{-1}(1-v_0)(1-v_1)$.

Throughout this paper, we will denote local rules of Boolean CA by g and their fuzzifications for the corresponding FCA by f. For any $\overrightarrow{i} \in \mathbb{Z}^d$, we will further denote $C^t(\overrightarrow{i})$ by $x^t_{\overrightarrow{i}}$, where $C^t : \mathbb{Z}^d \rightarrow [0,1]$ in the fuzzy case. When there is no confusion, as in the 1-dimensional case, the vector notation will be omitted.

A rule is said to *converge* to an homogeneous configuration $(\ldots p, p, p, \ldots)$ if, starting from an initial configuration $(\ldots, x^0_{i-1}, x^0_i, x^0_{i+1} \ldots)$ with $x^0_i \in (0,1)$ for all i, we have that $\forall \epsilon > 0\ \exists T$ such that $\forall t > T$ and $\forall i$: $|x^t_i - x^{t+1}_i| < \epsilon$. In this case, we will say that rule f *converges to* p.

In this paper, we are interested in the behaviour of *self-averaging rules*, a particular class of fuzzy CA. Self-averaging rules can be written as the weighted average of one of their variables with its negation as

(ii) note that, in the case of FCA, $max\{1,(a+b)\} = (a+b)$

follows (where x_i is the variable being averaged):

$$f(x_0, \cdots, x_{n-1}) f'(x_0, \cdots, x_{i-1}, x_{i+1}, \cdots, x_{n-1}) x_i +$$
$$(1 - f'(x_0, \cdots, x_{i-1}, x_{i+1}, \cdots, x_{n-1}))(1 - x_i).$$

Note that the function f could be the local function for a large 1-dimensional neighbourhood or for a neighbourhood of larger dimension. In the particular case of elementary FCA, we have: $f(x, y, z) = f'(y, z)x + (1 - f'(y, z))(1 - x)$ (analogously for variables y and z). For example, elementary rule 30 can be written as: $[(1 - y)(1 - z)]x + [(1 - y)z + y(1 - z) + yz](1 - x)$ and it is easy to see that, in this case, $f'(y, z) = (1 - y)(1 - z)$. A two-dimensional example would be the following rule: $f(x_{i,j}, x_{i,j+1}, x_{i+1,j}, x_{i,j-1}, x_{i-1,j}) = f_{40}(x_{i,j+1}, x_{i+1,j}, x_{i-1,j})x_{i,j} + f_{216}(x_{i,j+1}, x_{i+1,j}, x_{i-1,j})(1 - x_{i,j})$ where f_{40} and f_{216} are the complementary elementary functions. Note that in this example $x_{i,j-1}$ is a dummy variable since its value does not affect the result of the function. Table 1 contains all the elementary self-averaging rules where $\bar{x}$ indicates the value $(1 - x)$.

Rule	Equation
$f_{60}(x, y, z) \equiv f_{102}, f_{153}, f_{195}$	$(\bar{x})y + (x)\bar{y}$
$f_{90}(x, y, z) \equiv f_{164}$	$(\bar{x})z + (x)\bar{z}$
$f_{105}(x, y, z)$	$(\bar{x}y + x\bar{y})z + (\bar{x}\bar{y} + xy)\bar{z}$
$f_{150}(x, y, z)$	$(\bar{x}\bar{z} + xz)y + (\bar{x}z + x\bar{z})\bar{y}$
$f_{30}(x, y, z) \equiv f_{86}, f_{135}, f_{149}$	$(\bar{y}\bar{z})x + (\bar{y}z + y\bar{z} + yz)\bar{x}$
$f_{45}(x, y, z) \equiv f_{75}, f_{89}, f_{101}$	$(\bar{y}z)x + (\bar{y}\bar{z} + y\bar{z} + yz)\bar{x}$
$f_{106}(x, y, z) \equiv f_{120}, f_{169}, f_{225}$	$(\bar{x}\bar{y} + \bar{x}y + x\bar{y})z + (xy)\bar{z}$
$f_{154(x,y,z)} \equiv f_{166}, f_{180}, f_{210}$	$(\bar{x}y + \bar{x}\bar{y} + xy)z + (x\bar{y})\bar{z}$
$f_{108}(x, y, z) \equiv f_{201}$	$(\bar{x}\bar{z} + \bar{x}z + x\bar{z})y + (xz)\bar{y}$
$f_{156(x,y,z)} \equiv f_{198}$	$(\bar{x}\bar{z} + \bar{x}z + xz)y + (x\bar{z})\bar{y}$
$f_{54}(x, y, z) \equiv f_{147}$	$(\bar{x}\bar{z})y + (\bar{x}z + x\bar{z} + xz)\bar{y}$
$f_{57}(x, y, z) \equiv f_{99}$	$(\bar{x}z)y + (\bar{x}\bar{z} + x\bar{z} + xz)\bar{y}$

Table 1: Self-averaging elementary fuzzy CA rules.

3 Elementary Self-Averaging Rules

3.1 Convergence

In (2), we showed that from an initial configuration on $(0, 1)$ all self-averaging rules will converge to $\frac{1}{2}$. We are now interested in their asymptotic behaviour when the initial values are in the closed interval $[0, 1]$. In (18), Mingarelli proved the convergence to $\frac{1}{2}$ for some of the rules considered here for finite configurations in quiescent backgrounds (i.e., entirely consisting of cells in state 0s). In other words, we would like to see to what extent the presence of continuous values influences the dynamics of these rules. In fact, for some self-averaging rules we will see convergence to $\frac{1}{2}$ even with a single initial fuzzy value. Similar results hold for rules 60 and 90. We first state two simple lemmas.

Lemma 1 *Given* $x \in (0, 1)$ *then* $\alpha x + (1 - \alpha)(1 - x) \in (0, 1)$ *for all* $\alpha \in [0, 1]$.

PROOF Without loss of generality, assume $x \leq 1-x$ then $x \leq \alpha x + (1-\alpha)(1-x) \leq 1-x$:

$$\alpha x + (1-\alpha)(1-x) \leq \alpha(1-x) + (1-\alpha)(1-x) = (1-x) < 1,$$

and

$$\alpha x + (1-\alpha)(1-x) \geq \alpha x + (1-\alpha)x = x > 0.$$

∎

Lemma 2 *Given $\alpha \in (0,1)$ and $x \in [0,1]$ then $|(\alpha x + (1-\alpha)(1-x)) - \frac{1}{2}| < |x - \frac{1}{2}|$.*

PROOF Without loss of generality, assume $x < 1-x$ and let $x = \frac{1}{2} - \epsilon$ for some $\epsilon \in [0, \frac{1}{2}]$. Then $1 - x = \frac{1}{2} + \epsilon$. So $|x - \frac{1}{2}| = \epsilon$ and we need to show that $|(\alpha x + (1-\alpha)(1-x)) - \frac{1}{2}| < \epsilon$.

$$\begin{aligned} |(\alpha x + (1-\alpha)(1-x)) - \frac{1}{2}| &= |\alpha(\frac{1}{2} - \epsilon) + (1-\alpha)(\frac{1}{2} + \epsilon)) - \frac{1}{2}| \\ &= |\frac{1}{2} + \epsilon - 2\alpha\epsilon - \frac{1}{2}| \\ &= |\epsilon - 2\alpha\epsilon| \\ &= |1 - 2\alpha|\epsilon \\ &< \epsilon \end{aligned}$$

∎

The following convergence theorem holds for self-averaging rules which are permutive in every variable.

Theorem 1 *A single non-binary value in the initial configuration is sufficient to force convergence to $\frac{1}{2}$ for rules f_{105} and f_{150}.*

PROOF We assume that we have one non-binary value in our initial configuration at x_0^0. First note that, for rules 105 and 150, we can write these equations as self-averaging rules of any of their three variables. So the equations for x_{-1}^0, x_0^0, and x_1^0 can all be written in the $\alpha x + (1-\alpha)(1-x)$ where $\alpha \in \{0,1\}$ and $x \in (0,1)$. Then by Lemma 1 all three of these values must be fuzzy. Thus for any given cell i, $x_i^t \in (0,1)$ for all $t \geq |i|$ and the convergence follows from (2). ∎

We now turn to rule 90. In this case a single fuzzy value does not force the entire CA to converge to $\frac{1}{2}$, but it makes half of the cells converge. We can show that for total convergence we need two strategically placed fuzzy values.

Theorem 2 *Given rule 90 and an initial configuration $(\cdots, x_{-1}^0, x_0^0, x_1^0, \cdots)$ with one i such that value x_i^0 is in $(0,1)$, every other value converges to $\frac{1}{2}$ along a diagonal.*

PROOF First note that if x_i^t is in $(0,1)$ then x_{i-1}^{t+1} and x_{i+1}^{t+1} are in $(0,1)$. This follows from Lemma 1 and the following equations: $x_{i-1}^{t+1} = \bar{x}_{i-2}^t x_i^t + x_{i-2}^t \bar{x}_i^t$, and $x_{i+1}^{t+1} = \bar{x}_i^t x_{i+2}^t + x_i^t \bar{x}_{i+2}^t$. Furthermore, the sequence x_i^t, x_{i+1}^{t+1}, x_{i+2}^{t+2},... converges to $\frac{1}{2}$ by Lemma 2, noting that we can assume that the value of $|1-2\alpha|$ in the proof will be less than $|1 - 2x_i^t|$. Now assume, renumbering if necessary, that our initial fuzzy value was x_0^0. Then for any i, $x_i^{|i|}$ is in $(0,1)$ from the equations above, hence forms part of a convergent sequence as described. ∎

Theorem 3 *Given an initial configuration $X = (\cdots, x^0_{-1}, x^0_0, x^0_1, \cdots)$ with at least one even number i and one odd number j such that the values x^0_i and x^0_j (that is, 2 values that are an odd number of spaces apart) are in $(0, 1)$, rule 90 converges to $\frac{1}{2}$.*

PROOF Assume that the even value is 0. Then as in the proof above, at time $t = 1$, x^1_{-1} and x^1_1 will both be in $(0, 1)$. At $t = 2$, x^2_{-2}, x^2_0 and x^2_2 will all have fuzzy values. Continuing on in this way, we see that for any i x^t_i will have a fuzzy value for all $t \geq i$ such that the party of t and i are the same. Similarly, if we also have a fuzzy value with odd index, for all i there exists a T such that x^t_i has a fuzzy value whenever $t > T$ and i and t have opposite parity. Hence all values must converge to $\frac{1}{2}$. ∎

Note that in a circular CA of odd length, a single fuzzy value would be sufficient for rule 90 to converge to $\frac{1}{2}$ because every cell can be seen as being an even distance from every other cell; with a circular CA of even length the results of Theorem 3 would apply.

For rule 60, the results are a little different. With only a finite number of non-binary values, we can only show that rule 60 converges to $\frac{1}{2}$ to the right of all such values.

Theorem 4 *Given fuzzy rule f_{60} and at least one non-binary value in the initial configuration at x^0_i, then for all $j > i$ x^t_j will converge to $\frac{1}{2}$ as $t \to \infty$.*

PROOF By Lemma 1, x^t_i will remain in $(0, 1)$ for all t. Now since x^t_i is the weighting factor in the calculation of x^{t+1}_{i+1}, for all $t > 0$ x^t_{i+1} will be a fuzzy value. In fact, for any $j > i$, for any $t \geq j - i$, x^t_j will be a fuzzy value and by Lemma 2, they will all be converging to $\frac{1}{2}$ since the value of $|1 - 2\alpha|$ is decreasing at each iteration. ∎

Corollary 1 *Given fuzzy rule f_{60} and an initial configuration such that for all i there exists a $j < i$ such that x^0_j is non-binary, then this configuration will converge to $\frac{1}{2}$ everywhere. In particular, given a single non-binary value in a circular CA, all values will converge to $\frac{1}{2}$.*

Again note that in a circular CA, since there is no notion of left and right, a single value is enough to force convergence everywhere.

3.2 Behaviour around the Fixed Point

We are now interested in the way in which each of these rules converges to $\frac{1}{2}$. In this section, we show that if we are close enough to the point of convergence, we can determine if the output, $f(x, y, z)$ will be greater than or less than $\frac{1}{2}$ based on the individual values of x, y, and z and their relationship to $\frac{1}{2}$. In the range for which this relationship is fixed, we can write the results as a truth table with greater than and less than symbols. If we then interpret the less than symbols as 0s and the greater than symbols as 1s, we can deduce the equivalent binary behaviour of these rules as they approach $\frac{1}{2}$.

In (2), we began the discussion of the behaviour of the self-averaging rules as they approached the point of convergence. We now conclude it by describing how they fluctuate according to an elementary CA rule. These rules can be divided into two categories: those where the variable averaged is y can be shown to be all the elementary additive rules, and in (3) it has been shown that such rules are self-oscillating, that is, the truth table of their behaviour in proximity to their fixed point can be constructed from the truth table for the rule itself simply by replacing 0s with "<" and 1s with ">". The remaining rules behave as an entirely new rule in the proximity of $\frac{1}{2}$, a shift, an inversion or a combination of both. We sketch the proof for two of these rules, f_{108} and f_{156}; the corresponding behaviour for the others is reported in Table 3.

To begin, observe that if x and y are in $(1 - \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$, then xy, $\bar{x}y$, $x\bar{y}$, $\bar{x}\bar{y}$ are less than $\frac{1}{2}$.

Lemma 3 *$\alpha\beta + \bar{\alpha}\bar{\beta}$ is greater than $\frac{1}{2}$ if and only if both β and α are greater than $\frac{1}{2}$ or both are smaller.*

x	y	z	$f_{108\|156}$		x	y	z	g_{204}
$<$	$<$	$<$	$<$		0	0	0	0
$<$	$<$	$>$	$<$		0	0	1	0
$<$	$>$	$<$	$>$		0	1	0	1
$<$	$>$	$>$	$>$		0	1	1	1
$>$	$<$	$<$	$<$		1	0	0	0
$>$	$<$	$>$	$<$		1	0	1	0
$>$	$>$	$<$	$>$		1	1	0	1
$>$	$>$	$>$	$>$		1	1	1	1

Table 2: Rules 108 and 156: fluctuations around $\frac{1}{2}$

Theorem 5 *Rule f_{108} converges to $\frac{1}{2}$, its fluctuations around $\frac{1}{2}$ obeying Boolean rule g_{204}.*

PROOF We recall rule 108 in the form of a self-averaging rule:

$$(\bar{x}\bar{z} + \bar{x}z + x\bar{z})y + (xz)\bar{y}.$$

If we let $\alpha = \bar{x}\bar{z} + \bar{x}z + x\bar{z}$ and $\beta = y$, then from Lemma 3 $f(x, y, z)$ is greater than $\frac{1}{2}$ if $y > \frac{1}{2}$ and $\bar{x}\bar{z} + \bar{x}z + x\bar{z} > \frac{1}{2}$ or $y < \frac{1}{2}$ and $\bar{x}\bar{z} + \bar{x}z + x\bar{z} < \frac{1}{2}$. But $\bar{x}\bar{z} + \bar{x}z + x\bar{z} > \frac{1}{2}$ if and only if $xz < \frac{1}{2}$. From the observation above, for $x, z \in (1 - \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$, xz is always less than $\frac{1}{2}$ so $\bar{x}\bar{z} + \bar{x}z + x\bar{z}$ is always greater than $\frac{1}{2}$. Hence, under these conditions on x, z, $f_{108}(x, y, z) > \frac{1}{2}$ if and only if $y > \frac{1}{2}$. In other words, rule 108 fluctuates around $\frac{1}{2}$ obeying Boolean rule 204, $g(x, y, z) = y$. ∎

Analogously we obtain:

Theorem 6 *Rule f_{156} converges to $\frac{1}{2}$, its fluctuations around $\frac{1}{2}$ obeying Boolean rule g_{204}.*

Rule	behaviour around $\frac{1}{2}$
f_{60}	g_{60}
f_{90}	g_{90}
f_{105}	g_{105}
f_{150}	g_{150}

Rule	behaviour around $\frac{1}{2}$
f_{30}	$g_{15}(x, y, z) = \bar{x}$
f_{45}	$g_{15}(x, y, z) = \bar{x}$
f_{106}	$g_{170}(x, y, z) = z$
f_{154}	$g_{170}(x, y, z) = z$
f_{108}	$g_{204}(x, y, z) = y$
f_{156}	$g_{204}(x, y, z) = y$
f_{54}	$g_{51}(x, y, z) = \bar{y}$
f_{57}	$g_{51}(x, y, z) = \bar{y}$

Table 3: CA-like behaviour around the fixed point of self-averaging elementary fuzzy CA rules (the rules equivalent under conjugation, reflection, or both are not indicated).

3.3 Observations

Let us now reconsider the Boolean rules in light of the asymptotic behaviour of their fuzzy equivalents. We have seen that the auto-fluctuating rules continue to exhibit the same behaviour as their binary equivalents even as they converge. But what about the other rules? Can their convergent behaviour be seen in anyway in their corresponding Boolean rules? In fact, testing has shown that the same convergent behaviour as we have described here does occur. Line by line comparisons of Boolean rules 30, 45, 106, 154, 108, 156, 54, and 57 with the Boolean equivalent of the asymptotic functions of their fuzzy rules (that is, rules $\bar{x}$, $\bar{x}$, z, z, y, y, $\bar{y}$, and $\bar{y}$, respectively) show that they agree with these rules over 84% of the time, starting from random initial configurations.

Consider rules 106 and 154 illustrated in Figure 1. There is clearly, in both of them, a strong component of the rule z to which f_{106} and f_{154} converge. Rule 57 in Figure 2 appears to converge to z as well. However, this rule will sometimes have strong z components and sometimes strong x components. In fact, what it appears to be converging towards is a grid pattern ($\ldots 10101010 \ldots$) with "errors". The behaviour of such a pattern under rules x $\bar{y}$ and z is identical. Anomalies in the grid of the form 010010 are mapped to 101001 which appears to obeying $f(x, y, z) = x$ but can equally be seen as an error or exception to the rule $f(x, y, z) = \bar{y}$, while anomalies of the form 101101 map to 011010 which can again be either z or $\bar{y}$ with errors. Although rules 30 and 45 appear to be almost completely random, from the testing described above we observe that they obey $\bar{x}$ (i.e., $x_i^t = \bar{x^t}$) more than the 84.80 and 84.92 percent of the time, respectively, which is considerable. Rules 108 and 156, by contrast, can easily be seen to converge quickly to y with some errors.

4 General Self-Averaging Rules

In this section, we extend the results of the previous section to larger neighbourhoods and higher dimensions. In particular, we will give conditions under which generalized self-averaging rules will converge to $\frac{1}{2}$ everywhere and will give some indication of how a single fuzzy value will affect the system asymptotically. We will then describe the asymptotic oscillation of certain easily described subsets of the more general case.

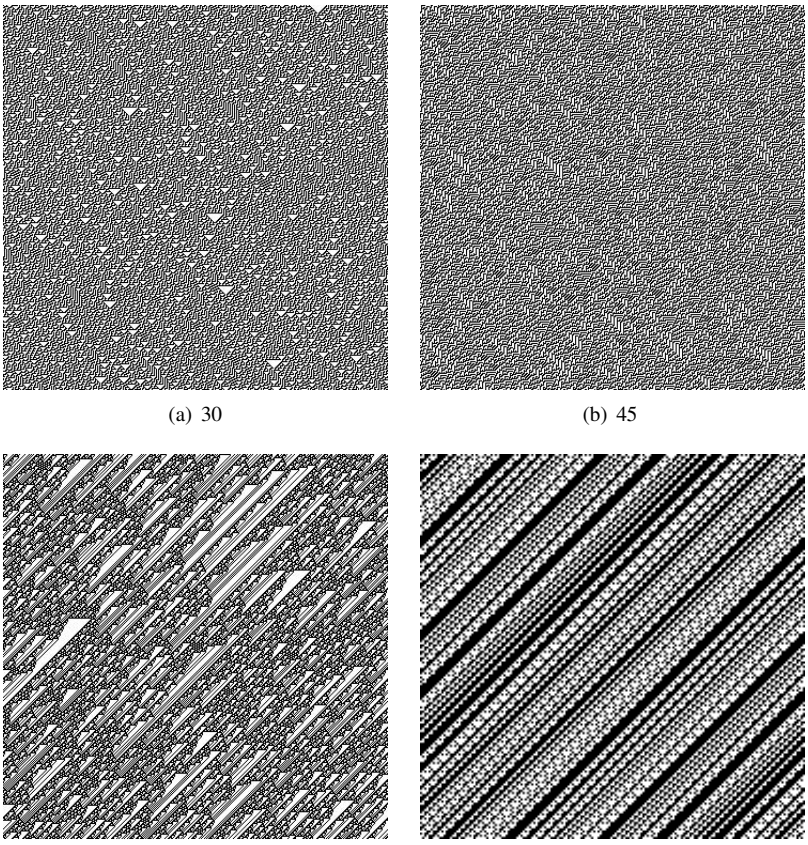
4.1 Convergence

In this section, we extend the proofs of convergence to self-averaging rules of any neighbourhood size or dimension.

Theorem 7 *Given initial configurations in* $(0, 1)$*, all self-averaging rules converge to* $\frac{1}{2}$.

PROOF Considering any of the self-averaging rules, we can see that for the weights to be precisely equal to 0 or 1, we must have values equal to 0 and 1 in the calculation. With initial configuration in $(0, 1)$, this will never happen by Lemma 1. Thus by Lemma 2, all values must approach $\frac{1}{2}$. Since the weight factor α is a sum of products of values which are approaching $\frac{1}{2}$, $|1 - 2\alpha|$ is bounded away from zero and one, hence all values must converge to $\frac{1}{2}$. ∎

How a fuzzy value will propagate depends on how many of its neighbours it affects. One way for a single fuzzy value to infect an entire system is if it is used non-trivially in the calculations of all the values in its von Neumann neighbourhood of range 1. This would imply that all of the values in its von Neumann neighbourhood are in the neighbourhood N of the cellular automaton and that they are not dummy variables.



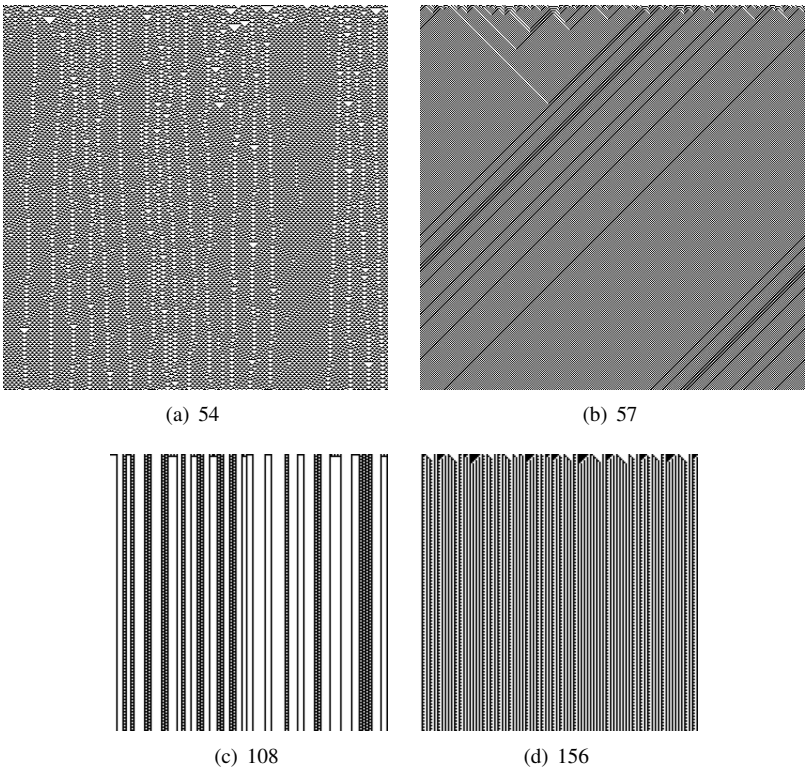
(a) 30 (b) 45

(c) 106 (d) 154

Figure 1: Boolean evolution of self-averaging rules: 30, 45,106,154.

Theorem 8 *Assume a self-averaging rule has a single non-binary value in its initial configuration. Further assume that the neighbourhood of a cell includes the von Neumann neighbourhood and that the function is non-trivially permutive in the von Neumann neighbourhood. Then the entire configuration will converge to* $\frac{1}{2}$.

PROOF Assume, without loss of generality, that $x_{\overrightarrow{0}}$ is in $(0, 1)$. Then at time $t = 1$, every cell in the von Neuman neighbourhood of range 1 of $x_{\overrightarrow{0}}$ will be in $(0, 1)$ by Lemma 1 since the local rule is permutive in $x_{\overrightarrow{0}}$. Inductively, at time t, the von Neumann neighbourhood of range t will be fuzzy. Given any $x_{\overrightarrow{i}}$, it is in the von Neumann neighbourhood of range T of $x_{\overrightarrow{0}}$ for some T. So at time $T+1$ its von Neumann neighbourhood of range 1 will consist entirely of fuzzy value. Hence by Lemma 2, $x^t_{\overrightarrow{i}}$ is converging to $\frac{1}{2}$ for $t > T + 1$. ∎



(a) 54 (b) 57

(c) 108 (d) 156

Figure 2: Boolean evolution of self-averaging rules: 54, 57, 108, 156.

We would expect one-sided convergence if only $\frac{1}{2}$ of the range 1 neighbourhood was used non-trivially and a multi-dimensional rule 60 effect, that is, a fixed pattern of every other cell converging to $\frac{1}{2}$, if the local rule used the cell itself and the cells in its range 2 neighbourhood that are not in the range 1 neighbourhood. We obtain rule 90 from the cells in range 1 except for the centre cell. In multiple dimensions, this causes every other cell to converge to $\frac{1}{2}$ along "diagonals", or, thought of another way, at every other time step.

4.2 Behaviour around the Fixed Point

As far as possible, we now generalize the results observed for the elementary self averaging rules.

First, recall that additive rules can be defined as the XORs of several variables or their negations. We can extend this definition to fuzzy rules by identifying additive rules with their fuzzifications. With additive rules, exactly half the entries in the truth table are equal to 1 and for all variables x_i, there are exactly as many terms in x_i for which the function f is equal to one as there are terms in $\bar{x}_i$ for which f is equal to one.

Theorem 9 *All additive rules are self-averaging rules exhibiting self-oscillations.*

PROOF That additive rules are self-averaging follows from the definitions. Re-numbering if necessary, we can write any additive rule $f(x_0, \cdots, x_{n-1})$ as

$$\begin{aligned} f(x_0, \cdots, x_{n-1}) &= \alpha(x_0, \cdots, x_{n-2}) \oplus x_{n-1} \\ &= (1 - \alpha(x_0, \cdots, x_{n-2}))x_{n-1} \\ &\quad +\alpha(x_0, \cdots, x_{n-2})(1 - x_{n-1}) \end{aligned}$$

which is clearly a self-averaging rule.

We prove the self-oscillations by induction on the number of variables. From (3), we know that this is true for functions in 3 or fewer variables. We assume that the hypothesis is true for n variables, and show that it must be true for $n+1$ variables. Consider the additive function f in $n+1$ variables. Re-numbering if necessary, we can write f as

$$f(x_0, \cdots, x_n) = \alpha(x_0, \cdots, x_{n-1})x_n + \overline{\alpha(x_0, \cdots, x_{n-1})}\bar{x}_n.$$

Now in order for f to be additive, α must also be additive. We know that f will be greater than $\frac{1}{2}$ when both x_n and α are greater than $\frac{1}{2}$, or when both are lesser. But by induction, α exhibits self-oscillations hence will be greater than $\frac{1}{2}$ precisely when its Boolean function would evaluate to 1. Under these conditions, when x_n is equal to 1, $g(x_0, \cdots, x_n) = 1$. The proof follows in the same way for $\alpha < \frac{1}{2}$. ∎

At the other extreme, we have rules where the value being self-averaged appears only once either directly or negated. These rules can easily be shown to converge towards that variable or its negation, whichever appears more often.

Theorem 10 *Given a Boolean self-averaging rule* $g(y_0, \cdots, y_{n-1}) = \alpha_g(y_0, \cdots, y_{i-1}, y_{i+1}, \cdots, y_{n-1})y_i + \overline{\alpha_g(y_0, \cdots, y_{i-1}, y_{i+1}, \cdots, y_{n-1})}\bar{y}_i$ *with weight function* α_g *such that there is only one element* $y \in \{0,1\}^{n-1}$ *such that* $\alpha_g(y) = 0$*, then the fuzzification* f *of* g *converges to a shift.*

PROOF Let $\gamma = (\frac{1}{2})^{\frac{1}{n-1}}$ and consider the interval $(1-\gamma, \gamma)$. If all variables x_i are on this interval then so are $\bar{x}_i$. Furthermore, the product of up to $n-1$ such variables is less than $[(\frac{1}{2})^{\frac{1}{n-1}}]^{n-1} = \frac{1}{2}$. Thus, when the entire configuration is on this interval, f is greater than $\frac{1}{2}$ precisely when x_i is. Hence it fluctuates as the shift $g(y_0, \cdots, y_{n-1}) = y_i$. ∎

Note that as the neighbourhood size grows, the interval on which the function behaves as a shift also grows.

For functions in between these two extremes, the extent to which they behave as a shift will depend on how unbalanced the weighting factors are.

5 Conclusions

In this paper we have concentrated on a class of fuzzy rules called self-averaging rules, whose asymptotic behaviour has previously been studied when all initial values were fully fuzzy (i.e., they all belonged to the open interval $(0,1)$). In this paper we have started to look at their behaviour when the initial values are in $[0,1]$. The presence of the extremes of the interval generally has a strong impact on the asymptotic behavior of a CA and a complete study of this impact will be the subject of future investigation. We have

then described the asymptotic fluctuations of this class of rules (with fully fuzzy initial configurations) around their fixed point.

In the second part of the paper, we have partially extended the results of (2) to CA in any dimension and any neighbourhood, showing that, in this case also, fully fuzzy configurations converge to $\frac{1}{2}$ and giving sufficient conditions for a single fuzzy value to cause the entire CA to converge to $\frac{1}{2}$. We have also partially generalized the results on the asymptotic fluctuations for this class of rules for higher dimensions and larger neighbourhoods. Further work is necessary to better understand how to fully generalize the fluctuating properties of these rules.

Acknowledgements

This work was partially supported by NSERC.

References

[1] A. I. Adamatzky. Hierarchy of fuzzy cellular automata. *Fuzzy Sets and Systems*, (62):167–174, 1994.

[2] H. Betel and P. Flocchini. Fluctuations of fuzzy cellular automata around their convergence point. In *Proc. of the International Symposium on Nonlinear Theory and its Applications, Japan*, 2009.

[3] H. Betel and P. Flocchini. On the relationship between boolean and fuzzy cellular automata. In *Proc. of the 15th International Workshop on Cellular Automata and Discrete Complex Systems*, volume 252, pages 5–21, 2009.

[4] H. Betel and P. Flocchini. On the asymptotic behavior of circular fuzzy cellular automata. *Journal of Cellular Automata*, 2010. To appear. Preliminary version in Proc. 15th Int. Workshop on Cellular Automata and Discrete Complex Systems, 2009.

[5] N. Boccara and K. Cheong. Automata network epidemic models. In *Cellular Automata and Cooperative Systems*, volume 396, pages 29–44. Kluwer, 1993.

[6] G. Cattaneo, P. Flocchini, G. Mauri, C. Quaranta-Vogliotti, and N. Santoro. Cellular automata in fuzzy backgrounds. *Physica D*, 105:105–120, 1997.

[7] G. Cattaneo, P. Flocchini, G. Mauri, and N. Santoro. Fuzzy cellular automata and their chaotic behavior. In *Proc. International Symposium on Nonlinear Theory and its Applications (NOLTA)*, volume 4, pages 1285–1289. IEICE, 1993.

[8] A.M. Coxe and C.A. Reiter. Fuzzy hexagonal automata and snowflakes. *Computers and Graphics*, 27:447–454, 2003.

[9] P. Flocchini and V. Cezar. Radial view of continuous cellular automata. *Fundamenta Informaticae*, 87(3):165–183, 2008.

[10] P. Flocchini, F. Geurts, A. Mingarelli, and N. Santoro. Convergence and aperiodicity in fuzzy cellular automata: revisiting rule 90. *Physica D*, 42:20–28, 2000.

[11] P. Flocchini and N. Santoro. The chaotic evolution of information in the interaction between knowledge and uncertainty. In R. J. Stonier and X. H. Yu, editors, *Complex Systems, Mechanism of Adaptation*, pages 337–343. IOS Press, 1994.

[12] K. Kaneko. *Theory and Application of Coupled Map Lattices*. John Wiley & Sons Ltd, 1993.

[13] G. Keller, M Kunzle, and T. Nowiki. Some phase transitions in coupled map lattices. *Physica D*, 59:39–51, 1992.

[14] C. G. Langton. Studying artificial life with cellular automata. In *Evolution, Games, and Learning*. North Holland, 1986.

[15] P. Maji. On characterization of attractor basins of fuzzy multiple attractor cellular automata. *Fundamenta Informaticae*, 86(1-2):143–168, 2008.

[16] P. Maji and P. P. Chaudhuri. Fuzzy cellular automata for modeling pattern classifier. *IEICE Transactions on Information and Systems*, 88(4):691–702, 2005.

[17] P. Maji and P. P. Chaudhuri. RBFFCA: A hybrid pattern classifier using radial basis function and fuzzy cellular automata. *Fundamenta Informaticae*, 78(3):369–396, 2007.

[18] A. Mingarelli. The global evolution of general fuzzy automata. *J. of Cellular Automata*, 1(2):141–164, 2006.

[19] A. Mingarelli. A study of fuzzy and many-valued logics in cellular automata. *J. of Cellular Automata*, 1(3):233–252, 2006.

[20] C. A. Reiter. Fuzzy automata and life. *Complexity*, 7(3):19–29, 2002.

[21] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, 1966.

[22] S. Wolfram. *Theory and Applications of Cellular Automata*. World Scientific, 1986.

[23] S. El Yacoubi and A. Mingarelli. Controlling the dynamics of the fuzzy cellular automaton rule 90. In *Proc. of 8th International Conference on Cellular Automata for Research and Industry (ACRI)*, pages 174–183, 2008.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 187–194

On entropy and Lyapunov exponents of dynamical systems generated by cellular automata

Maurice Courbage[1] and Brunon Kamiński[2†] and Jerzy Szymański[2‡]

[1] *Laboratoire Matière et Systèmes Complexes (MSC), Université Paris 7 - Diderot, Case 7056, Bâtiment Condorcet, porte 718A, 10, rue Alice Domon et Léonie Duquet, (France), E-mail: maurice.courbage@univ-paris-diderot.fr*
[2] *Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Chopina 12/18, 87-100 Toruń, (Poland), E-mail: bkam@mat.uni.torun.pl, jerzy@mat.uni.torun.pl*

The paper presents a new proof of the inequality between entropy and Lyapunov exponents given by Shereshevsky (1992) in the ergodic case.

1 Introduction

In [4] Shereshevsky associated to every dynamical system generated by a cellular automaton (CA-system) two remarkable real functions, called Lyapunov exponents, describing the dynamics of the system. One can consider these functions as analogues of Lyapunov exponents for smooth dynamical systems.

The main result of [4] contains an interesting inequality giving the connection between entropy and the Lyapunov exponents of a CA-system. One can look at this result as an analogue of the well knows Ruelle inequality in the theory of smooth dynamical systems.

The goal of our paper is to give a complete proof of the above inequality in the ergodic case. The motivation of our work are some gaps in the original proof in [4]. The main tools applied by us are some ideas of Tisseur ([5]) and the Ornstein-Weiss theorem being a generalization of the Breiman-McMillan-Shannon theorem.

2 Definitions and auxiliary results

Let $X = S^{\mathbb{Z}}$, $S = \{0, 1, \ldots, p-1\}$, $p \geq 2$ and let $\mathcal{B}$ be the σ-algebra generated by cylindric sets. We equip X with the distance d defined as follows (cf. [4])

$$d(x,y) = \begin{cases} 0 & \text{if} \quad x = y, \\ 2 & \text{if} \quad x_0 \neq y_0, \\ \exp(-N(x,y)) & \text{if} \quad x \neq y, x_0 = y_0 \end{cases}$$

†Partially supported by Polish MNiSzW grant N N201 384834
‡Partially supported by Polish MNiSzW grant N N201 384834

where $N(x,y)=\sup\{n\geq 0;\ x_i=y_i,\ |i|\leq n\},\ x,y\in X$.

We denote by σ the shift transformation of X and by f the automaton transformation of X generated by an automaton local rule F, i.e.

$$(\sigma x)_i=x_{i+1},\ \ (fx)_i=F(x_{i-r},\ldots,x_{i+r}),\ \ i\in\mathbb{Z},$$
$$F:S^{2r+1}\longrightarrow S,\ \ r\in\mathbb{N}.$$

For any $p,q\in\mathbb{Z}$, $p\leq q$ and $x\in X$ we denote by $\widetilde{F}(x_{p-r},\ldots,x_{q+r})$ the concatenation

$$\widetilde{F}(x_{p-r},\ldots,x_{q+r})=F(x_{p-r},\ldots,x_{p+r})F(x_{p+1-r},\ldots,x_{p+1+r})\ldots F(x_{q-r},\ldots,x_{q+r}).$$

It is obvious that

$$f(x)(p,q)=f(x)_pf(x)_{p+1}\ldots f(x)_q=\widetilde{F}(x_{p-r},\ldots,x_{q+r}).$$

By an interval in $\mathbb{Z}$ we mean a set which consists of all integers which belong to an interval in $\mathbb{R}$.

Let $I\subset\mathbb{Z}$ be an interval and let $x=(x_i),\ y=(y_i)\in X$. We shall write $x=y(I)$ if $x_i=y_i$, $i\in I$.

Let $x\in X$ and $s,p,q\in\mathbb{Z}$ be such that $p\leq q$. Following Shereshevsky ([4]) we put

$$W_s^+(x)=\{y\in X;y=x(s,+\infty)\},$$
$$W_s^-(x)=\{y\in X;y=x(-\infty,-s)\}.$$
$$C_p^q(x)=\{y\in X;\ y=x(p,q)\}.$$

For a given $n\geq 1$ one defines

$$\widetilde{\Lambda}_n^{\pm}(x)=\inf\left\{s\geq 0;f^n(W_0^{\pm}(x))\subset W_s^{\pm}\left(f^nx\right)\right\}$$

and

$$\widetilde{l}_n^{\pm}(x)=\inf\left\{s\geq 0;\forall_{0\leq i\leq n}\ f^i(W_0^{\pm}(x))\subset W_s^{\pm}\left(f^ix\right)\right\}$$

It is clear that

$$\widetilde{l}_n^{\pm}(x)=\max\left(\widetilde{\Lambda}_1^{\pm}(x),\ldots,\widetilde{\Lambda}_n^{\pm}(x)\right).$$

We put

$$\Lambda_n^{\pm}(x)=\sup_{j\in\mathbb{Z}}\widetilde{\Lambda}_n^{\pm}(\sigma^jx),\qquad l_n^{\pm}=\sup_{j\in\mathbb{Z}}\widetilde{l}_n^{\pm}(\sigma^jx).$$

It easy to see that

$$l_n^{\pm}(x)=\max\left(\Lambda_1^{\pm}(x),\ldots,\Lambda_n^{\pm}(x)\right).$$

It is shown in [4] that the limits

$$\lambda^{\pm}(x)=\lim_{n\to\infty}\frac{1}{n}\Lambda_n^{\pm}(x)$$

exist a.e. and they are f and σ-invariant and integrable.

The limit λ^+ (resp. λ^-) is called the right (left) Lyapunov exponent of f.

It is easy to show that

$$0\leq\lambda^{\pm}(x)\leq r$$

and

$$\lambda^{\pm}(x)=\lim_{n\to\infty}\frac{l_n^{\pm}(x)}{n}.$$

Lemma 1 *For any natural numbers n, p, i such that $n \geq 0$, $0 \leq i \leq n$, $p \geq 2r$ and $x \in X$ we have*

$$f^i\left(C^{p+l_n^-(x)}_{-p-l_n^+(x)}(x)\right) \subset C^p_{-p}\left(f^i x\right).$$

Proof: We shall use in the sequel the abbreviation $l_n^{\pm} = l_n^{\pm}(x)$.

Let $y \in C^{p+l_n^-(x)}_{-p-l_n^+(x)}(x)$. We have to show that

$$\forall_{0\leq i\leq n}\ f^i(y) = f^i(x)(-p, p). \tag{1}$$

It is clear that the sets

$$W^+_{-p-l_n^+}(x) \cap W^-_{-p-l_n^-}(y), \qquad W^+_{-p-l_n^+}(y) \cap W^-_{-p-l_n^-}(x)$$

consist of single elements. Let us denote them by z and w, respectively. Thus we have

$$z = x\left(-p - l_n^+, +\infty\right), \qquad z = y\left(-\infty, p + l_n^-\right) \tag{2}$$

and

$$w = x\left(-\infty, p + l_n^-\right), \qquad w = y\left(-p - l_n^+, +\infty\right). \tag{3}$$

From (2) and (3) it follows that

$$f^i(z) = f^i(x)(-p, +\infty) \tag{4}$$

and

$$f^i(w) = f^i(x)(-\infty, p) \tag{5}$$

for every $0 \leq i \leq n$.

Indeed, applying the formula (cf. [2])

$$\sigma^a W_c^{\pm}\left(\sigma^b x\right) = W^{\pm}_{c\mp a}\left(\sigma^{a+b}x\right),\ a, b, c \in \mathbb{Z}, x \in X.$$

We get

$$\begin{aligned}
&f^i\left(W^+_{-p-l_n^+(x)}(x)\right) = \sigma^{p+l_n^+(x)} f^i\left(W_0^+\left(\sigma^{-p-l_n^+(x)}x\right)\right)\\
&\subset \sigma^{p+l_n^+(x)} W^+_{\widetilde{l_n^+}\left(\sigma^{-p-l_n^+(x)}x\right)}\left(\sigma^{-p-l_n^+(x)} f^i x\right)\\
&\subset \sigma^{p+l_n^+(x)} W^+_{l_n^+(x)}\left(\sigma^{-p-l_n^+(x)} f^i x\right)\\
&= W^+_{-p}\left(f^i x\right), \quad 0 \leq i \leq n.
\end{aligned}$$

This means that (4) is satisfied. Similarly we show the inclusion

$$f^i\left(W^-_{-p-l_n^-(x)}(x)\right) \subset W^-_{-p}\left(f^i x\right), \quad 1 \leq i \leq n.$$

what gives (5).

Now we start to show (1) by induction w.r. to $i \in \{0, \ldots, n\}$.
The property (1) is obviously true for $i = 0$ because $l_n^{\pm} \geq 0$.
Let us now suppose that for some $0 \leq i \leq n-1$ it holds

$$\forall_{0 \leq k \leq i} f^k(y) = f^k(x)(-p, p). \tag{6}$$

We shall show that

$$f^{i+1}(y) = f^{i+1}(x)(-p, p). \tag{7}$$

First we shall prove that (6) implies

$$\begin{aligned} \forall_{0 \leq k \leq i}\, f^k(z) &= f^k(y)(-p - r(i+1-k), r), \\ f^k(w) &= f^k(y)(-r, p + r(i+1-k)). \end{aligned} \tag{8}$$

Let us prove the first equality. The proof of the second one is analogous.
We argue by induction w.r. to $k \in \{0, \ldots, i\}$. The validity of (8) for $k = 0$ follows at once from the inequalities $p \geq r, l_n^-, l_n^+ \geq 0$.

Suppose now that

$$\forall_{0 \leq k \leq i-1}\, f^k(z) = f^k(y)(-p - r(i+1-k), r). \tag{9}$$

We shall show that

$$f^{k+1}(z) = f^{k+1}(y)(-p - r(i-k), r). \tag{10}$$

We have

$$\begin{aligned} &f^{k+1}(z)(-p - r(i-k), r) = \widetilde{F}\left(f^k(z)(-p - r(i+1-k), 2r)\right) = \\ &= \widetilde{F}\left(f^k(z)(-p - r(i+1-k), r)\right) \widetilde{F}\left(f^k(z)(-r+1, 2r)\right). \end{aligned} \tag{11}$$

The assumption (9) tells that

$$f^k(z) = f^k(y)(-p - r(i+1-k), r).$$

We claim that

$$\widetilde{F}\left(f^k(z)(-r, 2r)\right) = \widetilde{F}\left(f^k(y)(-r+1, 2r)\right).$$

Indeed, the equality (4) gives

$$f^i(z) = f^i(x)(-p, +\infty)$$

for any $0 \leq i \leq n$. Hence in particular

$$f^k(z) = f^k(x)(-p, +\infty)$$

and so, since $p \geq r$, we get

$$f^k(z) = f^k(x)(-r+1, 2r). \tag{12}$$

We have $k \leq i-1$, i. e. $k+1 \leq i$ and therefore applying (6) we have

$$f^{k+1}(y) = f^{k+1}(x)(-p, p).$$

Hence by (12) and $p \geq r$ we get

$$\begin{aligned} &\widetilde{F}\left(f^k(z)(-r+1,2r)\right) = \widetilde{F}\left(f^k(x)(-r+1,2r)\right) = \\ &= f^{k+1}(x)(1,r) = f^{k+1}(y)(1,r) = \\ &= \widetilde{F}\left(f^k(y)(-r+1,2r)\right). \end{aligned} \tag{13}$$

Therefore returning to (11) we have

$$\begin{aligned} &f^{k+1}(z)(-p-r(i-k),r) = \\ &= \widetilde{F}\left(f^k(y)(-p-r(i+1-k),r)\right)\widetilde{F}\left(f^k(y)(-r+1,2r)\right) = \\ &= f^{k+1}(y)(-p-r(i-k),0)f^{k+1}(y)(1,r) = \\ &= f^{k+1}(y)(-p-r(i-k),r) \end{aligned}$$

which gives (10) and so (8).

Substituting $k = i$ in (8) we get

$$\begin{aligned} f^i(z) &= f^i(y)(-p-r,r), \\ f^i(w) &= f^i(y)(-r,p+r). \end{aligned} \tag{14}$$

Now we shall finish the proof showing (7), i.e. the equality

$$f^{i+1}(y) = f^{i+1}(x)(-p,p).$$

We have

$$\begin{aligned} & f^{i+1}(y)(-p,p) = \widetilde{F}\left(f^i(y)(-p-r,p+r)\right) = \\ = \;& \widetilde{F}\left(f^i(y)(-p-r,r)\right)\widetilde{F}\left(f^i(y)(-r+1,p+r)\right) = \\ \overset{(14)}{=}\;& \widetilde{F}\left(f^i(z)(-p-r,r)\right)\widetilde{F}\left(f^i(w)(-r+1,p+r)\right) = \\ = \;& f^{i+1}(z)(-p,0)f^{i+1}(w)(1,p) = \\ \overset{(4),(5)}{=}\;& f^{i+1}(x)(-p,0)f^{i+1}(x)(0,p) = f^{i+1}(x)(-p,p) \end{aligned}$$

which gives the desired result. □

Theorem 1 *For any Borel probability measure invariant w.r. to σ and f and ergodic with respect to f or σ it holds*

$$h_\mu(f) \leq \left(\lambda^+ + \lambda^-\right) h_\mu(\sigma).$$

Proof: Let G be the set of all $x \in X$ for which the limits

$$\lim_{n\to\infty} \frac{l_n^{\pm}(x)}{n} = \lambda^{\pm}(x) \tag{15}$$

exist and the functions λ^+, λ^- are constant. It follows from our above considerations that $\mu(G) = 1$.

Let $p \in \mathbb{N}$, $p \geq 2r$ and $\delta > 0$ be arbitrary. By (15) there exists $N = N_\delta$ such that

$$\frac{l_n^{\pm}(x)}{n} \leq \lambda^{\pm} + \delta \tag{16}$$

for $n > N$.

Now Lemma implies

$$f^i \left(C^{p+l_n^-(x)}_{-p-l_n^+(x)}(x) \right) \subset C^p_{-p}\left(f^i x\right), \tag{17}$$

for any $0 \leq i \leq n$.

We put

$$\lambda_n^{\pm}(\delta) = \left[\left(\lambda^{\pm} + \delta\right) n\right] + 1, \quad \varepsilon_p = e^{-p}.$$

Let

$$B_n(f, x, \varepsilon_p) = \left\{ y \in X;\ d\left(f^k y, f^k x\right) < \varepsilon_p,\ 0 \leq k \leq n \right\}.$$

Our aim is to show the inclusion

$$B_n(f, x, \varepsilon_p) \supset C^{p+\lambda_n^-(\delta)}_{-p-\lambda_n^+(\delta)}(x),\ x \in G. \tag{18}$$

It follows from (16) that $l_n^{\pm}(x) \leq \lambda_n^{\pm}(\delta)$, $n > N$.

Let $y \in C^{p+\lambda_n^-(\delta)}_{-p-\lambda_n^+(\delta)}(x)$. Hence by (17) we get

$$\begin{aligned} f^k(y) &\in f^k \left(C^{p+\lambda_n^-(\delta)}_{-p-\lambda_n^+(\delta)}(x) \right) \subset \\ &\subset f^k \left(C^{p+l_n^-(x)}_{-p-l_n^+(x)}(x) \right) \subset \\ &\subset C^p_{-p}\left(f^k x\right), \quad 0 \leq k \leq n,\ \ n > N. \end{aligned}$$

This means that

$$\left(f^k y\right)_m = \left(f^k x\right)_m, \quad -p \leq m \leq p$$

and so

$$N\left(f^k y, f^k x\right) \geq p,$$

i.e.

$$d\left(f^k y, f^k x\right) \leq e^{-p}, \quad 0 \leq k \leq n.$$

In other words $y \in B_n(f, x, \varepsilon_p)$ which proves (18).

Let now

$$A_n = A_n(\delta, p) = \left\{ m \in \mathbb{Z};\ -p - \lambda_n^+(\delta) \leq m \leq p + \lambda_n^-(\delta) \right\}$$

and let P denote the zero-time partition of X.

It is clear that

$$C^{p+\lambda_n^-(\delta)}_{-p-\lambda_n^+(\delta)}(x) = \left(\bigvee_{m \in A_n} \sigma^m P \right)(x),\ n > N.$$

Therefore (18) implies

$$\mu\left(B_n(f,x,\varepsilon_p)\right) \geq \mu\left(\left(\bigvee_{m\in A_n}\sigma^m P\right)(x)\right). \tag{19}$$

Now we shall check that $(A_n) = (A_n(\delta,p))$ is a Følner sequence for any $\delta > 0$, $p \geq 1$.

Indeed, for any $g \in \mathbb{Z}$, $g \geq 0$ we have

$$(g + A_n) \cap A_n = \left\{m \in \mathbb{Z};\ -p + g - \lambda_n^+(\delta) \leq m \leq p + \lambda_n^-(\delta)\right\}$$

for any $n \geq 0$.

Hence for such n we have

$$\frac{|(g + A_n) \cap A_n|}{|A_n|} = \frac{(2p - g + \lambda_n^-(\delta) + \lambda_n^+(\delta) + 1)}{\left(2p + \lambda_n^-(\delta) + \lambda_n^+(\delta)\right)}$$

and so

$$\lim_{n\to\infty}\frac{|(g + A_n) \cap A_n|}{|A_n|} = 1.$$

In the same way one can show that the above is true for $g < 0$. Hence (A_n) is a Følner sequence. Applying the Ornstein-Weiss theorem (cf. [3]) and the fact that P is a generator for σ we get

$$\lim_{n\to\infty}\frac{1}{|A_n|}\log\mu\left(\left(\bigvee_{m\in A_n}\sigma^m P\right)(x)\right) = h_\mu(P,\sigma) = h_\mu(\sigma) \tag{20}$$

for $x \in G$. Therefore applying (19) and (20) we have

$$\overline{\lim_{n\to\infty}}\left(-\frac{1}{n}\log\mu\left(B_n(f,x,\varepsilon_p)\right)\right) \leq \left(\lambda^+ + \lambda^- + 2\delta\right) h_\mu(\sigma),$$

$p \geq 1$, $\delta > 0$ and $x \in G$.

Now taking the limits as $\delta \to 0$ and $p \to \infty$ we get

$$h_\mu(f,x) \leq \left(\lambda^+ + \lambda^-\right) h_\mu(\sigma),$$

and applying the Brin-Katok formula (cf. [1]) we obtain the desired inequality

$$h_\mu(f) \leq \left(\lambda^+ + \lambda^-\right) h_\mu(\sigma).$$

□

References

[1] M. Brin and A. Katok. On local entropy. *Geometric dynamics (Rio de Janeiro, 1981), Lecture Notes in Math.*, 1007:30–38, 1983.

[2] M. Courbage and B. Kamiński. Space-time directional Lyapunov exponents for cellular automata. *J. Stat. Phys.*, 124(6):1499–1509, 2006.

[3] D. S. Ornstein and B. Weiss. The Shannon-McMillan-Breiman theorem for a class of amenable groups. *Israel J. Math.*, 44(1):53–60, 1983.

[4] M. A. Shereshevsky. Lyapunov exponents for one-dimensional cellular automata. *J. Nonlinear Sci.*, 2(1):1–8, 1992.

[5] P. Tisseur. Cellular automata and Lyapunov exponents. *Nonlinearity*, 13(5):1547–1560, 2000.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 195–208

Relative Partial Reversibility of Elementary Cellular Automata

Pedro P.B. de Oliveira[1,2] and Rodrigo Freitas[2]

Universidade Presbiteriana Mackenzie
[1]*Faculdade de Computação e Informática &* [2]*Pós-Graduação em Engenharia Elétrica*
Rua da Consolação 896, Consolação
01302-907 São Paulo, SP - Brazil

We address the notion of partial reversibility of cellular automata rules. The elementary space is the focus of all analyses, which rely upon the individual initial configurations for which a given rule is reversible or not, under periodic boundary conditions. These are represented in what is defined herein as the reversibility pattern of the rule. By lexicographically sorting the elementary space according to this construct, the space becomes partitioned into 45 classes of reversibility equivalence, where their positions provide an indication of their relative reversibility degree. The analysis of some of the classes unveil very intriguing properties of their rule members. Preliminary ideas towards modelling the reversibility degree of the classes are discussed. It is tempting that the results can be generalised to other rule spaces and that they can be used towards defining an absolute measure of partial reversibility degree of a rule, but there are difficulties ahead that require further considerations.

Keywords: Cellular automata, reversible rules, partially reversible rules, reversibility pattern, elementary space, α parameter, Z parameter.

1 Introduction

A well-studied property of some cellular automata (CAs) is reversibility, that is, the property possessed by some rules of having their temporal evolution regenerated backward in time, regardless of the original initial configuration, by running the inverse rule of the original [Toffoli and Margolus(1990)]. Reversibility in CAs is such a well characterised concept that it has been possible to derive many fundamental results associated with it, such as the undecidability of the property for CAs in dimensions larger than 1 [Kari(2005)], algorithms to enumerate all reversible one-dimensional CAs [Boykett(2004)], etc.

But what if one would be willing to define a notion of *partial* reversibility, such that CA rules could then be compared in terms of their *degree* of reversibility? To provide some first insights into issues involving this question is the motivation of this paper.

In order to go about that, our analyses rely upon the individual initial conditions for which a given CA rule is reversible or not. And for the sake of simplicity, we restrict the analyses to the elementary CA space, i.e., the set of one-dimensional cellular automata, with 2 states per cell, and next-nearest neighbours

(neighbourhood size of 3 cells), that comprises 256 rules. But the insights given from this CA space can be readily extended to further one-dimensional CA spaces.

One motivation for addressing the notion of partial reversibility is the possibility of probing the notion for its own sake, looking at possible ways to measure it, possible theoretical models that can account for the measures, its relations to other CA properties, such as dynamical behaviour, etc. And because there cannot be an algorithm for establishing the reversibility of CAs in dimensions larger than 1, the possibility of devising a was to 'measure' the partial reversibility degree of a rule would certainly be useful, in that it would help pointing at specific rules that would stand a chance of being reversible.

Reversibility is a property found in only a handful of rules of any space; for instance, for the elementary rules, only 6 of them are reversible, out of the 256 possible rules. So, from an applications perspective, the idea of defining partial reversibility may also be appealing; for instance, just as reversibility has had a role in conceiving algorithms for encryption (as in [Seredynski et al.(2004)Seredynski, Pienkosz, and Bouvry]), one might also think of devising algorithms based on the notion of partially reversible rules which, as for one advantage, might lead to a much larger set of rule options to use in such algorithms.

The focus of this paper is on the *relative* partial reversibility of the rules, that is, the possibility of discriminating rules in terms of their relative degree of reversibility. Given the ill-defined nature of partial reversibility, it seems natural to study first the relative notion of the concept, before an attempt to devise an absolute notion. This is precisely what is carried out in the paper.

The next section discusses a way to check the reversibility of one-dimensional cellular automata through their pre-images, as well as the relations of this feature to two parameters that can be defined for a CA. The following section introduces the concept of *reversibility pattern* of cellular automata, which is the basis for the subsequent discussions on partial reversibility. Some concluding remarks are made at end, pointing at the following paths to be pursued in the work. All presentation is couched in terms of the elementary space.

2 Reversibility Checking in Cellular Automata

2.1 Direct testing of the pre-images

A cellular automaton (CA) is reversible if there exists an *inverse* rule to the latter, which can undo the temporal evolution backwards of the original CA, thus leading back to its initial configuration (IC), whichever it is.

In order to establish a rule to be reversible, all initial configurations must have one and only one pre-image, the irreversible rules thus entailing some configurations to have multiple or no pre-image at all. Figure 1 illustrates the situation, by displaying the basins of attraction of the elementary rule 38, with lattice size 6, where the dots represent the 2^6 possible lattice configurations, and the edges their corresponding pre-images; notice that while the configurations in cyclic circuits are reversible, the others are not.

One procedure for checking whether a given one-dimensional CA rule is reversible or not is described in [Wolfram(2002)]. In this procedure, all initial configurations must be tested up to a certain maximum lattice size n_{max}, and what it does is checking whether each IC has one and only one pre-image. Although it has been proven that the required upper bound is $n_{max} = k^{2r}(k^{2r}-1)+2r+1$, with k representing the number of cell states and r being the neighbourhood radius, [Wolfram(2002)] states that there is empirical evidence for the necessity of considering only the smaller value $n_{max} = k^{2r}$. In fact, all empirical tests

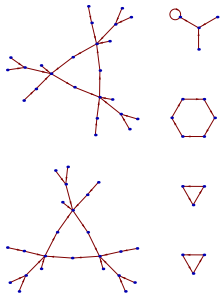


Figure 1: Basins of attraction for elementary rule 38 with lattice size 6.

we carried out, involving reversible rules from various one-dimensional spaces, are in tune with such a statement.

Noticing that the procedure above may require significant computational effort, one might think about possible less demanding alternatives, which includes the ideal possibility of their being directly drawn from the rule table of a CA. Two such parameters (Z and α) are discussed next, since both, by definition and by their own way, are related to the pre-images of a CA rule and, therefore, may somehow be related also to reversible rules.

2.2 Z-parameter

The reverse algorithm proposed in [Wuensche and Lesser(1992)] computes all pre-images, if any, of a global configuration, or determines whether the current state has no pre-image at all, i.e., if it is a Garden-of-Eden (GoE) configuration.

The Z parameter is directly derived from this algorithm. As a partial pre-image is being built from a given configuration, Z provides the probability of the next unknown state in that pre-image being uniquely determined. As a result, it indicates the density of GoE configurations or the density of pre-images from the basin of attraction of the corresponding rule. Thus, higher values of Z imply lower density of GoE configurations and less 'bushy' basins of attraction; in other words, higher values of Z imply fewer pre-images from a configuration and longer paths of subsequent pre-images in the basin of attraction of the configuration.

The Z parameter is composed by Z_{left} and Z_{right}. Z_{left} is obtained by running the reverse algorithm from left to right in the partial pre-image, while Z_{right} is obtained by running the same algorithm from right to left. Z is defined as the largest value among the latter two. As such, the larger the value of Z, the 'more' reversible is the rule, the largest value 1 happening for the reversible rules.

2.3 Parameters α and α^p

Motivated by the analysis of the relationship between the pre-images of a global configuration and the reversibility of cellular automata, the α parameter was defined in [Schranko and de Oliveira(2010)].

Consider a CA with k states per cell (defined over the alphabet $\Sigma = \{0, 1, \ldots, k-1\}$), neighbourhood with m cells, and transition rule f. Now, letting $\mathcal{B}_m$ represent the set of (basic) blocks of size m, in Σ, then the set of pre-images of $\mathcal{B}_m$ is $\mathcal{B}_{2m-1}$ and its size is given $|\mathcal{B}_{2m-1}| = k^{2m-1}$. Accordingly, in its simplest form the αparameter can be defined as follows:

$$\alpha = \frac{1}{k^{2m-1}} \sum_{i \in T} i \ , \quad \text{where } T = \bigcup_{b \in \mathcal{B}_m} \{|f^{-1}(b)|\} \tag{1}$$

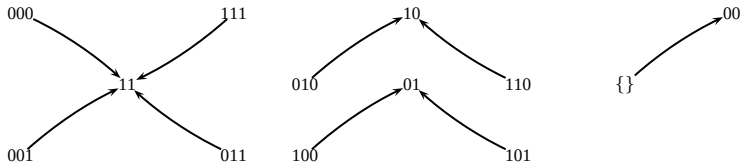


Figure 2: Pre-images of all blocks, for the binary CA rule 11, with 2 cells in the neighbourhood [Schranko and de Oliveira(2010)].

A clarification of the definition can be obtained by calculating α for the binary CA rule 11 with $m = 2$; Figure 2 illustrates the situation. As such, $\mathcal{B}_2 = \{00, 01, 10, 11\}$ and $|\mathcal{B}_{2*2-1}| = |\mathcal{B}_3| = 2^3 = 8$. The application of f^{-1} to each block $b \in \mathcal{B}_2$ entails $|f^{-1}(00)| = 0$, $|f^{-1}(01)| = 2$, $|f^{-1}(10)| = 2$ and $|f^{-1}(11)| = 4$; then, by uniting the values of $|f^{-1}(b)|$, one of the values 2 is removed, so that the sum of the remaining values becomes $\sum_{i \in \{0,2,4\}} i = 6$, thus leading to $\alpha = 6/8 = 0.75$.

The definition in [Schranko and de Oliveira(2010)] is actually more general, in that it can account for any hyper-rectangular neighbourhood. But the essential is that they show the existence of a relation between α, the distribution of pre-images into basic blocks and reversibility, so that if a cellular automaton is reversible, then the number of pre-images for each basic block is equally distributed, and α takes its minimum value. Naturally, this is in tune with the fact that the distribution of all pre-images of surjective CA rules is balanced.

Notice that, in the definition of α, the predecessor blocks associated with the basic blocks of a rule can be seen as the required pre-images to generate the basic blocks, in non-periodic boundary conditions. In analogy with this, we define herein a new parameter, denoted α^p, which is a slight variation of α, with the only difference being that α^p relies upon the actual pre-images of the possible configurations (or arbitrary blocks), in *periodic* boundary condition. In doing so, α^p also becomes somehow related to reversibility, and this is the issue for present purposes.

Since α relies upon the pre-image of each block, one could also question the effects of using not only the latter, but also the pre-images of the latter, or the pre-images of the pre-images of the latter, etc. As such, we can generalise the definition of α in terms of the level (or 'order', so to speak) of the pre-images considered, which leads to the notion of an i-th order α, where the original definition of α becomes in fact the 1st order α, or α_1, and correspondingly, α_2, α_3, and so on. These alternatives are also considered in the analyses below. But notice that, although it might also be possible to refer to the i-th order α^p, this is not useful, since the values of α^p are the same for all orders, because all pre-images of any order are exactly the same.

3 Towards Partial Reversibility

3.1 Reversibility pattern of a rule

In order to define partial reversibility, every cyclic initial configuration of a particular rule should be examined from the point of view of its number of pre-images.

Hence, a concept is proposed here, namely, *reversibility pattern* of a rule, which is in the basis for the characterisation of partial reversibility of the rule, and relies on testing all the initial configurations of the lattice, from the minimum size $n = 1$, up to the maximum n_{max}. Our approach follows the procedure in [Wolfram(2002)], that checks whether a given one-dimensional CA rule is reversible or not, as mentioned earlier. Since our tests rely on the elementary CA rules, for which $k = 2$ and $r = 1$, it turns out that, while the theoretical upper bound is $n_{max} = 15$, the empirical value is $n_{max} = 4$.

For example, the reversibility pattern for elementary rule 2, as worked out through the empirical $n_{max} = 4$, is ($\{2, 2, 2, 2, 8\}$, $\{1, 1, 1, 5\}$, $\{4\}$, $\{2\}$), where the four multisets refer, respectively, to the lattice sizes n equals to 4, 3, 2 and 1, and every number in a multiset corresponds to the number of pre-images of a given configuration of size n. Hence, $\{2, 2, 2, 2, 8\}$ represents that, out of the 2+2+2+2+8=16 possible 4-bit-long ICs, 4 of them are not reversible because each one has 2 pre-images, 1 is not reversible because it has 8 pre-images, and the remaining 8 ICs (not explicitly appearing in the multiset) are not reversible because they are GoE configurations; similarly, $\{1, 1, 1, 5\}$ represents that, out of the 1+1+1+5=8 possible 3-bit-long ICs, 3 of them are reversible because each one has a single pre-image, 1 is non-reversible because it has 5 pre-images, and the remaining 4 ICs (not explicitly appearing in the multiset) are not reversible because they are GoE configurations; and so on. Naturally, for present purposes it only really makes sense to consider lattice sizes which are at least the same as the neighbourhood size; here we display all values of n just for the sake of simplicity.

As a consequence, it can be argued that the elementary rule 2 has a small partial reversibility degree, or, it is partially reversible, since some of its initial configurations are reversible and some are not.

But, how can one state that a given rule is 'more' (partially) reversible than another? Let us consider the elementary rules below and their corresponding reversibility patterns, also derived from $n_{max} = 4$. By comparing the reversibility patterns of rules 62 and 44 it seems reasonable to consider the latter as more reversible than the former, since they only differ in the information associated with lattice size $n = 3$, which favours rule 44. But the comparison is not as clear, when comparing rules 151 and 223; after all, while the former has 12 reversible ICs and the latter only 11, the former is totally non-reversible for lattice size $n = 3$ which is not the case for the latter. So, which criteria should be taken into account so that a comparison can be made is an issue that requires further investigation. This is what is done next, by analysing the consequences of sorting the elementary rules in two distinct ways.

62 = ($\{1, 1, 1, 1, 2, 2, 2, 2, 2, 2\}$, $\{1, 1, 1, 2, 3\}$, $\{2, 2\}$, $\{2\}$)
44 = ($\{1, 1, 1, 1, 2, 2, 2, 2, 2, 2\}$, $\{1, 1, 1, 1, 1, 1, 2\}$, $\{2, 2\}$, $\{2\}$)
151 = ($\{1, 1, 1, 1, 1, 1, 1, 1, 1, 6\}$, $\{3, 5\}$, $\{1, 1, 2\}$, $\{2\}$)
223 = ($\{1, 1, 1, 1, 1, 1, 10\}$, $\{1, 1, 1, 5\}$, $\{1, 1, 2\}$, $\{2\}$)

3.2 Relative partial reversibility in the elementary space

For what follows, the elementary space is accounted for not in terms of its 256 rules, but in reference to its 88 classes of dynamical equivalence [Wolfram(2002)], each one being referred to by the rule with the

smallest rule number in the class. Naturally, such an approach suffices since every rule in a class has the same reversibility pattern.

Given the difficulties raised above in respect to determining the relative partial reversibility among rules, here we look at the entire space, trying to obtain insights for the issue. The idea is to derive the reversibility pattern of all rules in the elementary CA space and order them lexicographically, from (supposedly) the least to the most reversible rules. As argued in the first section, only after devising a reasonable rationale for relatively measuring partial reversibility should one think to step forward towards an absolute measure, which is not within the scope of the present paper.

Even for a relative measure, based upon the notion of reversibility pattern, one decision that has to be made concerns the two values of n_{max}, discussed earlier. After all, they were derived as constraints that should be respected when establishing the (full) reversibility of a rule; hence, their usage might simply not be applicable for present purposes. So, a better understanding of the effects of using one value of n_{max} or the other, or even another, is definitely one of the issues that need being addressed here.

Along this line, the first ordering scheme to be looked at herein is the standard, direct lexicographical order entailed by the reversibility pattern of a rule. Following this scheme, each part that makes up the reversibility pattern is individually and subsequently taken, starting at the first (associated with the largest lattice size), and used as the basis for the relative sorting. For the sake of easy reference, let us denote this approach as the *direct* sorting scheme of the reversibility patterns.

For instance, with the empirical $n_{max} = 4$, elementary rules 1, 11 and 27, would be lexicographically sorted, in relative terms as follows:

$$
\begin{array}{lcl}
1 & = & (\{1, 1, 1, 1, 1, 11\}, \{1, 7\}, \{1, 3\}, \{1, 1\}) \\
11 & = & (\{1, 1, 1, 1, 1, 2, 2, 2, 2, 3\}, \{1, 1, 2, 2, 2\}, \{1, 3\}, \{1, 1\}) \\
27 & = & (\{1, 1, 1, 1, 1, 2, 2, 2, 2, 3\}, \{\ 1, 1, 1, 1, 1, 1, 1, 1\}, \{1, 3\}, \{1, 1\})
\end{array}
$$

On the other hand, the second relative sorting scheme considered here is based upon the reversibility pattern of a rule considered as a whole, by merging its the parts into a single set, therefore, with no distinction among the individual lattice sizes. This type of sorting is referred to below as the *absolute* scheme.

Considering once again the elementary rules 1, 11 and 27, their absolute lexicographical sorting is shown below. Notice that it is more appealing now that rule 27 should be assumed to be more partially reversible than the others, since it has the largest number of reversible initial configurations.

$$
\begin{array}{lcl}
1 & = & \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 7, 11\} \\
11 & = & \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3\} \\
27 & = & \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3\}
\end{array}
$$

The examples above have all been with the smallest value of n_{max}. But in order to evaluate the effect of using the largest value, Table 1 depicts a comparison of the two. Notice that the implicit number of GoE configurations of each rule is not being taken into account.

Table 1: Direct and absolute lexicographical sortings of the elementary space, for the two upper-bound values of lattice size drawn from [Wolfram(2002)].

Direct Sorting		Absolute Sorting	
$n_{max} = 4$	$n_{max} = 15$	$n_{max} = 4$	$n_{max} = 15$
{0}	{0}	{0}	{0}
{90}	{105, 150}	{90}	{46}
{46}	{24}	{126}	{24}
{2, 8}	{10}	{46}	{36}
{126}	{46}	{24}	{126}
{36}	{12, 34}	{36}	{90}
{24}	{126}	{60}	{60}
{60}	{36}	{2, 8}	{10}
{5, 160}	{90}	{12, 34}	{12, 34}
{9, 130}	{60}	{10}	{2, 8}
{12, 34}	{2, 8}	{18, 72}	{11, 138}
{13, 162}	{11, 138}	{62, 110}	{4, 32}
{57, 156}	{4, 32}	{9, 130}	{1, 128}
{33, 132}	{1, 138}	{1, 128}	{43, 142}
{43, 142}	{43, 142}	{94, 122}	{29, 184}
{94, 122}	{29, 184}	{37, 164}	{18, 72}
{18, 72}	{18, 72}	{11, 13, 58, 78, 138, 162}	{13, 162}
{37, 164}	{13, 162}	{38, 44}	{5, 160}
{10}	{5, 160}	{5, 160}	{3, 19, 136, 200}
{62, 110}	{3, 19, 136, 200}	{4, 32}	{27, 172}
{38, 44}	{27, 172}	{30, 106}	{9, 130}
{1, 128}	{9, 130}	{23, 232}	{6, 40}
{11, 138}	{77, 178}	{22, 104}	{77, 178}
{27, 172}	{6, 40}	{33, 132}	{33, 132}
{4, 32}	{33, 132}	{26, 74}	{23, 232}
{23, 232}	{23, 232}	{25, 152}	{28, 50, 56, 76}
{77, 178}	{28, 50, 56, 76}	{14, 42}	{35, 140}
{30, 106}	{35, 140}	{6, 40}	{38, 44}
{14, 42}	{38, 44}	{54, 108}	{14, 42}
{58, 78}	{14, 42}	{3, 19, 136, 200}	{54, 108}
{28, 50, 56, 76}	{54, 108}	{73, 146}	{58, 78}
{26, 74}	{58, 78}	{27, 28, 50, 56, 57, 76, 156, 172}	{7, 168}
{35, 140}	{7, 168}	{35, 140}	{94, 122}
{41, 134}	{94, 122}	{43, 142}	{26, 74}
{3, 19, 136, 200}	{57, 156}	{41, 134}	{73, 146}
{25, 152}	{26, 74}	{77, 178}	{57, 156}
{73, 146}	{73, 146}	{45, 154}	{41, 134}
{45, 154}	{41, 134}	{7, 168}	{25, 152}
{22, 104}	{25, 152}	{105, 150}	{62, 110}
{6, 40}	{62, 110}	{29, 184}	{37, 164}
{29, 184}	{22, 104}	{15, 51, 170, 204}	{22, 104}
{7, 168}	{37, 164}		{105, 150}
{54, 108}	{30, 106}		{30, 106}
{105, 150}	{45, 154}		{45, 154}
{15, 51, 170, 204}	{15, 51, 170, 204}		{15, 51, 170, 204}

The first general observation is that the table shows that in the absolute sorting scheme the results

were quite different from those in the direct sorting scheme. In fact, notice that the list generated with $n_{max} = 4$ contains rule classes that have not appeared before, and that the list is shorter than the one with $n_{max} = 15$. Also, the length of the latter list (45 classes) is exactly the same size as both lists of the direct sorting scheme, all rule classes also being the very same, even though some of them placed at distinct positions.

Analysing the direct sorting scheme, Table 1 also makes it evident that the position of the set of (linear) rules $\{105, 150\}$ in both lists strikes as just too different from each other. In order to examine why that might have been so, if suffices to check the reversibility pattern of rule 105, with $n_{max} = 15$; one can readily observe that a clear pattern is generated for each lattice size, namely, that the rule is fully reversible for every lattice size not multiple of 3, and not reversible otherwise, in which case one quarter of the ICs have 4 pre-images each and the remaining three-quarters are GoE configurations; Table 2 illustrates the situation, with the smaller value $n_{max} = 6$, for the sake of concisiveness. This is an intriguing observation, with no obvious explanation and apparently not previously reported in the literature, that points at the validity and usefulness of the concept of reversibility pattern of a rule defined herein.

Table 2: Reversibility pattern of elementary rule 105, with maximum lattice size $n_{max} = 6$.

Lattice size	Reversibility pattern
6	{4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4}
5	{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
4	{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
3	{4, 4}
2	{1, 1, 1, 1}
1	{1, 1}

Comparing the two lists of the direct sorting scheme the consequence of using the distinct values of n_{max} becomes evident: almost all rule classes had their positions altered from one list to the other. This means that, while the upper-bounds defined in [Wolfram(2002)] are sufficient for establishing whether a rule is reversible or not, they do not equally apply to determining (relative) partial reversibility. In fact, this even questions the appropriateness of using the largest upper-bound, thus raising the possibility that another value of n_{max} should be used instead, or, in the worst case, of even questioning the existence at all of a truthful upper-bound.

In order to address these questions, Figure 3 was created, for all values of n_{max} from 1 to 18, plotting the positions of all rule classes from Table 1 that appear in the longer list of the absolute scheme (which are the same as those in the direct sorting scheme). The higher the position of a rule class, the more reversible it is. The positions refer to the absolute sorting scheme, so that all points plotted for every n_{max} are the result of computing the reversibility pattern for all lattice sizes from 1 to n_{max}. It is clear from the figure that, although there is no convergence for each and every rule class, the vast majority of them do converge, and the few of those that do not, at least become fairly stabilised, with a local fluctuation. Naturally, in order to know for sure what happens afterwards for rule classes such as 10 or 23, 232, the figure would have to be extended for further values of n_{max}.

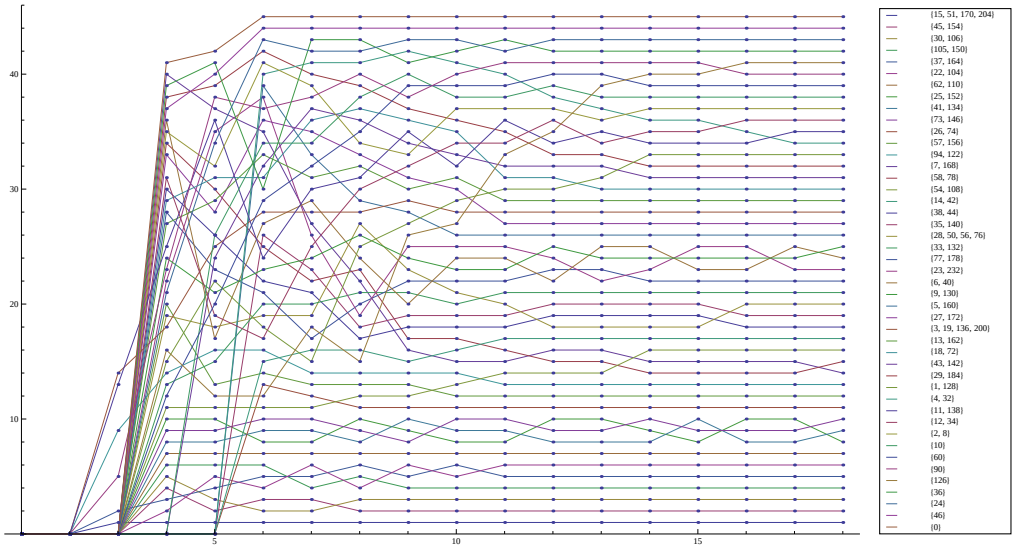


Figure 3: Position of each rule class for all lattice sizes n_{max} from 1 to 18, considering the absolute sorting scheme.

So, all the points above put into perspective suggest that the absolute sorting scheme seems to be more adequate to represent partial reversibility of rules than the other. And in fact, except in the explicit necessity of addressing the issue for a specific lattice size, there is no apparent reason to weigh the individual sizes, thus leading to their being taken into account as a single, whole ensemble.

3.3 Relating partial reversibility with CA-based parameter values

It would be useful to model partial reversibility, by resorting to a simple, computationally non-intensive method. As mentioned earlier, the one in [Wolfram(2002)] does not fit this criterion, so that it can really be thought of as providing the basis for an empirical measure of the quantity. Naturally, a good candidate for a model, as hinted at in Section 2, could be CA-based parameters whose values could be more easily worked out, including the desired possibility of drawing the values directly from the rule table of a CA.

But in order for one such parameter to be a real candidate, it is required that the partition of the elementary space induced by their values do not compromise the rule classes entailed by the lexicographical sorting scheme judged as the most adequate (the absolute scheme). In other words, any candidate parameter cannot split the rule classes that are obtained out of the absolute sorting scheme, regardless of their values. This is what is checked next.

The parameters at issue were primarily those discussed previously in the paper (Z, α, α_2, α_3 and α^p), but also others from the literature, according to [Oliveira et al.(2001)Oliveira, de Oliveira, and Omar], related to estimates of the dynamical behaviour of CA rules (λ, *Sensitivity*, *Neighbourhood Dominance*, *Absolute Activity* and *Absolute Propagation*).

With the dynamically-oriented parameters, for *Sensitivity* and λ only the class $\{3, 19, 136, 200\}$) is split. For the other parameters (*Neighbourhood Dominance*, *Absolute Activity* and *Absolute Propagation*) almost all classes are split; but this is not surprising since all these parameters show no similarity between the rule classes they induce in the elementary space and those derived from the sorting schemes.

With the other parameters, only α^p does not compromise the classes of the sorting schemes; however, the set of rule classes it induces is much smaller than the latter, meaning that it also has no similarity with the classes induced by the sorting schemes.

As for parameters, α, α_2 and α_3, they all show that only one rule class ($\{3, 19, 136, 200\}$ again) of the absolute sorting scheme is split. Finally, for parameter Z, two classes of the sorting schemes are split, $\{28, 50, 56, 76\}$ and, once again, $\{3, 19, 136, 200\}$. Table 3 shows the rule classes entailed from α^p, α and Z, respectively.

Table 3: Rule classes of the elementary space induced by α^p, α and Z.

Rules	α^p	Rules	α	Rules	Z
{15, 27, 29, 43, 45, 51 57, 77, 142, 154, 156, 170, 172, 178, 184, 204}	0.125	{15, 30, 45, 51, 60, 90, 105 106, 150, 154, 170, 204}	0.125	{0}	0
{10, 12, 24, 34, 36, 46, 58, 60, 78, 90}	0.25	{35, 43, 140, 142}	0.25	{1, 2, 4, 8, 32, 128}	0.25
{11, 13, 14, 25, 26, 28, 35, 37, 38, 42, 44, 50, 56, 74, 76, 138, 140, 152, 162, 164}	0.375	{29, 184}	0.3125	{**3**, 5, 6, 9, 10, 12, 18, 23, 24, 29, 33, 34, 36, 40, 43, 46, 72, 77, 126, 130, 132, **136**, 142, 160, 178, 184, 232}	0.5
{23, 105, 150, 232}	0.5	{27, 46, 57, 156, 172}	0.375	{**19**, 35, **50**, **76**, 140, **200**}	0.625
{3, 5, 7, 9, 19, 33, 41, 73, 130, 132, 134, 136, 146, 160, 168, 200}	0.625	{23, 77, 178, 232}	0.40625	{7, 11, 13, 14, 22, 25, 26, 27, **28**, 37, 38, 41, 42, 44, 54, **56**, 57, 58, 62, 73, 74, 78, 94, 104, 108, 110, 122, 134, 138, 146, 152, 156, 162, 164, 168, 172}	0.75
{2, 4, 6, 8, 18, 30, 32, 40, 54, 62, 72, 94, 106, 108, 110, 122}	0.75	{7, 168}	0.4375	{15, 30, 45, 51, 60, 90, 105, 106, 150, 154, 170, 204}	1
{0, 1, 22, 104, 126, 128}	1	{13, 28, 50, 56, 76, 162}	0.46875		
		{26, 41, 54, 58, 74, 78, 108, 134}	0.5		
		{11, 138}	0.53125		
		{38, 44, 62, 110}	0.5625		
		{9, 14, 22, 33, 37, 42, 104, 130, 132, 164}	0.59375		
		{6, 18, 40, 72}	0.625		
		{**19**, 73, 146, **200**}	0.65625		
		{**3**, 36, 126, **136**}	0.6875		
		{25, 94, 122, 152}	0.71875		
		{2, 5,8, 10, 160}	0.75		
		{12, 24, 34}	0.8125		
		{1, 128}	0.84375		
		{4, 32}	0.90625		
		{0}	1		

Another intriguing fact becomes apparent when attempting to identify a possible cause for why the rule class $\{3, 19, 136, 200\}$ is split so often. Taking elementary rule 3 as the representative of that rule class, its reversibility pattern with $n_{max} = 15$ shows a peculiar behaviour. Table 4 exemplifies the situation, where each column is related to the reversibility pattern associated with a given lattice size n. More precisely, for a given n, each cell in table displays the number of initial configurations of size n that have a certain number of pre-images. These two quantities are, respectively, the numbers appearing in each row, in the form: *the number of ICs*[*the number of pre-images*]. So, for example, with lattice size 15 (column 1), there are 1364 ICs with only one pre-image (row 1), 960 ICs with 2 pre-images each (row 2), etc.

Table 4: The number of initial configurations of size n that have a certain number of pre-images, for elementary rule 3.

Lattice size (n)														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1364[1]	841[1]	521[1]	324[1]	199[1]	121[1]	76[1]	49[1]	29[1]	16[1]	11[1]	9[1]	4[1]	1[1]	2[1]
960[2]	560[2]	325[2]	180[2]	99[2]	60[2]	36[2]	16[2]	7[2]	6[2]	5[2]	1[7]	1[4]	1[3]	
600[3]	350[3]	195[3]	108[3]	66[3]	40[3]	18[3]	8[3]	7[3]	6[3]	1[11]				
180[4]	91[4]	39[4]	18[4]	11[4]	5[4]	9[5]	8[5]	7[5]	1[18]					
375[5]	210[5]	117[5]	72[5]	44[5]	20[5]	9[8]	8[8]	1[29]						
195[6]	84[6]	39[6]	24[6]	11[6]	10[8]	9[13]	1[47]							
230[8]	126[8]	78[8]	48[8]	22[8]	10[13]	1[76]								
45[9]	21[9]	13[9]	6[9]	11[13]	10[21]									
90[10]	42[10]	26[10]	12[10]	11[21]	1[123]									
135[13]	84[13]	52[13]	24[13]	11[34]										
45[15]	28[15]	13[15]	12[21]	1[199]										
45[16]	28[16]	13[16]	12[34]											
90[21]	56[21]	26[21]	12[55]											
30[24]	14[24]	13[34]	1[322]											
15[25]	7[25]	13[55]												
30[26]	14[26]	13[89]												
60[34]	28[34]	1[521]												
15[39]	14[55]													
15[40]	14[89]													
15[42]	14[144]													
30[55]	1[843]													
15[89]														
15[144]														
15[233]														
1[1364]														

Now, let us take from the table, for every lattice size, the largest number of pre-images associated with any initial configuration, that is, the number of pre-images of the most non-reversible ICs for each lattice size; in the notation of the table, these are the ones having the form 1[*number*] for lattice sizes 2 to 15, and the 2[1] for size 1 (since this is the only possibility). As a consequence, the following numerical sequence is generated, from the smallest lattice size to the largest: 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364.

It turns out that these numbers constitute exactly the so-called Lucas numbers, a series of integer digits akin to the well-known Fibonacci sequence [Weisstein(2010)]; more specifically, the n-th term in the sequence above is exactly the n-th Lucas number, for n varying from 1 to 15. It is quite an interesting and intriguing fact to realise that the Lucas numbers seem to directly govern the amount of least reversible

initial configurations of every size, that is, those having the largest possible number of pre-images for the CA rule at issue.

Equally as striking is that the sequence of the number of reversible ICs is also governed by the Lucas numbers, although not as directly as before. In fact, considering s_n the number of the most reversible ICs with lattice size n, and L_n the n-th Lucas number, s_n is given by:

$$s_n = \begin{cases} L_0 & \text{if } n = 1 \\ L_n & \text{if } n > 1 \text{ and } n \text{ is odd} \\ L_n - 2 & \text{if } n > 1 \text{ and } n \text{ is even and } n/2 \text{ is odd} \\ L_n + 2 & \text{if } n > 1 \text{ and } n \text{ is even and } n/2 \text{ is even} \end{cases} \tag{2}$$

Lucas numbers have already appeared in the context of discrete dynamical systems, specifically, in recursive integer sequences [Wolfram(2002)]; however, as far as we know, no connection has been made so far in the literature involving Lucas numbers and the global behaviour of CA rules, or with elementary rules in particular.

4 Concluding Remarks

Here we studied the possibility of defining the notion of partial reversibility of an elementary CA rule. Our focus was on the relative comparisons between the rules. By creating a lexicographical sorting scheme of the rules we showed that it makes sense to point at a rule being more reversible than another.

Although how our results generalise to larger one-dimensional, binary CA spaces has not been attempted, it seems likely this can be done. In fact, a key motivation for the research has been the possibility of modelling partial reversibility, according to the lessons learnt from the elementary space, so that this can be applicable to other rule spaces.

The experiments carried out involving the absolute sorting scheme clearly showed that both upper bounds defined in [Wolfram(2002)] ($n_{max} = 4$ and $n_{max} = 15$, in the case of the elementary space) do not suffice for establishing the relative partial reversibility of a rule. Nevertheless, the value $n_{max} = 15$ seems acceptable when compared with $n_{max} = 18$, the largest value we computed; the ideal upper bound is unknown.

The experiments also allowed to consider the elementary rule space as partitioned into 45 classes of reversibility equivalence. Whether this can really be regarded as true in the limit of an infinite lattice size is something that cannot be asserted by now. Although the possibility is really tempting, a theoretical difficulty would have to be accommodated, derived from the fact that reversibility over cyclic configurations does not suffice for granting reversibility of a rule on an unrestricted lattice ([Kari(2005)]). And since this applies to any CA space, this same difficulty would have to be faced in order to generalise the notions addressed here, towards larger one-dimensional binary CA spaces and CA spaces with larger dimensions and/or larger number of states per cell.

By relating partial reversibility with CA-based parameter values, including those drawn from the rule tables, it is clear that Z and α (including its variations α^p, α_2, α_3) has some correlation with partial reversibility. Therefore, it may be possible to use them, combined somehow, as models of partial reversibility. Evaluating this idea is one of the steps ahead . In this direction, understanding the precise role of these parameters is a key ingredient, one indication being that α might possibly be considered a measure of the surjectivity degree of a CA.

Another issue also open for further investigation is to come up with a way to calculate the partial reversibility degree of an elementary rule. This has to be made so that we can forego the notion of relative partial reversibility in favour of the absolute notion.

A related conceptual question refers to the definition of the closest possible *partially* inverse rule that would correspond to a given partially reversible rule. This is a very ill-defined concept that should only be addressed after the previous questions have been very carefully looked at.

The usefulness of the notion we introduced here, of reversibility pattern, has been made evident with all the issues above but also because it allowed to unveil intriguing properties, apparently unknown so far, involving the partial reversibility of elementary rules 105 and 3, and the other members of their respective reversibility classes. And since we have not inspected other equivalence classes that display curious behaviour suggested from Figure 3, such as $\{10\}$ or $\{23, 232\}$, it will not be a surprise if further intriguing phenomena involving their partial reversibility do spring up. Nevertheless, the explicit consideration of the GoE configurations in the reversibility pattern of a rule is yet another issue that needs to be looked at.

Acknowledgements

We acknowledge the travel grant provided by Universidade Presbiteriana Mackenzie for attending the workshop, and thank very fruitful comments provided by the reviewers of our original manuscript.

References

[Boykett(2004)] T. Boykett. Efficient exhaustive listings of reversible one dimensional cellular automata. *Theor. Comput. Sci.*, 325(2):215–247, 2004. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2004.06.007.

[Kari(2005)] J. Kari. Theory of cellular automata: a survey. *Theor. Comput. Sci.*, 334(1-3):3–33, 2005. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2004.11.021.

[Oliveira et al.(2001)Oliveira, de Oliveira, and Omar] G. Oliveira, P. de Oliveira, and N. Omar. Definition and applications of a five-parameter characterization of one-dimensional cellular automata rule space. *Artificial Life*, 7(3):277–301, 2001.

[Schranko and de Oliveira(2010)] A. Schranko and P. P. B. de Oliveira. Relationships between local dynamics and global reversibility of multidimensional cellular automata with hyper-rectangular neighborhoods. *Unpublished manuscript*, 2010.

[Seredynski et al.(2004)Seredynski, Pienkosz, and Bouvry] M. Seredynski, K. Pienkosz, and P. Bouvry. Reversible cellular automata based encryption. In *Network and Parallel Computing (LNCS Series)*, volume 3222, pages 411–418, 2004.

[Toffoli and Margolus(1990)] T. Toffoli and N. H. Margolus. Invertible cellular automata: a review. *Phys. D*, pages 229–253, 1990.

[Weisstein(2010)] E. W. Weisstein. Lucas number. http://mathworld.wolfram.com/LucasNumber.html, 2010.

[Wolfram(2002)] S. Wolfram. *A new kind of science*. Wolfram Media, 2002.

[Wuensche and Lesser(1992)] A. Wuensche and M. Lesser. *The Global Dynamics of Cellular Automata*, volume Reference Vol 1 of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, 1992. IBSN 0-201-55740-1.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 209–220

Evolving Probabilistic CA Agents Solving the Routing Task

Patrick Ediger[1] and Rolf Hoffmann[1]

[1] *Technische Universität Darmstadt, FB Informatik, FG Rechnerarchitektur, Hochschulstr. 10, 64389 Darmstadt, Germany*

Given is a $2D$ field of $n \times n = N$ cells (communication nodes) with border. The goal was to solve the routing problem with N agents, each of them having the task to transport a message from a source to a target. This task is also known as multiple target searching. The whole agent system was modeled as a uniform cellular automaton. The agents shall have a probabilistic behavior in order to avoid deadlocks and livelocks. Three types of agents were defined: (1) FSM controlled agents (with an embedded finite state machine evolved by a genetic algorithm), (2) XY agents using the XY-routing technique, and (3) MXY agents (modified XY agents). To all agents' behaviors an optimal amount of randomness was added. It turned out that for $N = 256$ the best FSM agents solved the task within 84 generations, 14% faster than the MXY agents, and 31.5% faster than the XY agents. The added randomness was lowest (1.8%) for the FSM agents. Additional tests have showed that the number of generations can further be reduced by 17.5% for $N = 256$, using 420 additional empty cells in an enlarged field of size $(n + 11)^2$.

Keywords: CA Agents, Routing with Agents, Multiple-Target Searching, Multi-Agent Systems, Evolving Probabilistic Behavior, FSM Controlled Agents

1 Introduction

We are presenting a method that allows evolving the probabilistic behavior of moving CA agents in order to solve a given task, exemplified by the routing problem. The routing problem with agents can also be seen as a multiple target searching problem where each agent searches for its individual target. A moving CA agent is a set of CA rules modeling the agent's behavior within the CA paradigm. In particular, we are modeling agents that can decide upon their actions by the use of an embedded control automaton (finite state machine).

In order to communicate between processors on a chip an appropriate network has to be supplied. We assume that the communication is based on packets/messages transported from a source to a target (destination) processor. A lot of research has been carried out in order to find the best networks with respect to latency, throughput, fault tolerance, and so on. Instead of improving the known design principles we want to follow a novel approach based on agents that transport messages. Routing a message can be *deterministic* (unique path is taken) or *adaptive* (alternative links are selected during message passing). Our goal is to find an optimal adaptive routing technique using intelligent agents. Each agent selects dynamically on its own the links using a control algorithm that was evolved by a genetic procedure. We

will specialize our investigation to the $2D$ grid structure with cyclic connections, in which the processors and the network components are located at dedicated positions.

Related Work. A Cellular Automata (CA) based path planning algorithm in multi-agent systems has been proposed in Tavakoli et al. (2008), where many agents have to find the same target. Our investigation is related to this work, but a main difference in our task is that each agent has its own individual target. Target searching in agent systems has been researched in many variations: with moving targets in Loh and Prakash (2009); Goldenberg et al. (2003); Koenig et al. (2007), and in single-agent systems in Korf (1990). Here we restrict our investigation to stationary targets and multiple agents having only a local view. Adaptive routing algorithms with mobile agents have been presented in Caro and Dorigo (1997); Dhillon and Mieghem (2007) using software agents inspired by ant behavior. In contrast to these works, a simple finite state machine controls our agents, and the agents are intended to be implemented in hardware.

This contribution continues our preceding work on routing with agents on a $2D$ grid (Ediger and Hoffmann (2009, 2010)). In Ediger and Hoffmann (2009) four routing models with agents were proposed and compared (undirected agent, randomized undirected agent, directed agent, randomized directed agent). There it turned out, that many of the evolved directed agents were very reliable, but that deadlocks could not be avoided securely. In Ediger and Hoffmann (2010) the optimal spatial distribution of communication nodes was investigated. It turned out that one or two free nodes (buffers, spaces) between communication nodes were optimal with respect to the overall reachable communication time.

Other routing algorithms were investigated for regular $2D$ grid structures (mesh) including non-adaptive techniques, e. g., XY-routing in de Mello et al. (2004) and adaptive techniques, e. g., hot potato routing. In Busch et al. (2001) a *greedy*, *local* and *dynamic* hot potato algorithm is presented. Our technique produces adaptive routing algorithms that are local and not necessarily greedy. The test cases in this paper are all *static* according to Busch et al. (2001): "*all packets are injected at time zero*", but could also be applied to dynamic systems: "*nodes may inject packets into the network repeatedly over a long duration*" (Busch et al. (2001)).

In other former works, we investigated multi-agent systems in CA with different tasks, like the Creature's Exploration Problem in Halbach et al. (2006) or the All-to-All Communication task in Ediger and Hoffmann (2008b). In these investigations we used different methods of optimization like genetic programming (Komann et al. (2009)), genetic algorithms (Ediger and Hoffmann (2008a)), sophisticated enumeration (Halbach (2008)) and time-shuffling techniques (Ediger and Hoffmann (2008b)). A transactional CA model for multi-agent systems was developed in Spicher et al. (2009). In general our work is also related to works like: evolving optimal rules for CA (Sipper (1997); Sipper and Tomassini (1999)), detect centroids with marching pixels (Komann et al. (2007)), simulation of pedestrian behavior (Schadschneider (2008)) or traffic flow (Schadschneider and Schreckenberg (1993)).

The remainder of this paper is organized as follows. In Section 2 we explain the agents' task and the CA model in more detail. The method of evolving the probabilistic FSM controlled agents is described in Section 3. The performance of the evolved agents is compared against the XY-routing agents in Section 4. Section 5 concludes, and proposes further investigations.

2 The Task and the Modeling of the Agents

2.1 The Routing Task

Given is a $2D$ grid of $n \times n = N$ cells with border. A cell can dynamically be either of type EMPTY, OBSTACLE or AGENT. Obstacles are used to model the border, or they can be used to model broken

cells in a network. Each cell can be used as a source and/or the target of a message. Therefore, a cell can act as a communication node or processor. A *message transfer* is the transfer of one message from a source to a target. A set of messages shall be called *message set*. A *message set transfer* is the successful transfer of all the messages belonging to the set. The agents shall perform message transfers, the whole system we call *agent system*. Initially the agents are located at their source positions. Then they move to their targets. When an agent reaches its target, it is deleted. Thereby the number of moving agents is reduced until no agent is left. This event defines the end of the whole message set transfer. In order to simplify this investigation we constrained the problem:

- The number s of source cells is equal to the number d of target cells, and it is equal to the number k of agents, and it is equal to N: $k = s = d = N$. This means that initially an agent is placed in each cell of the array without spaces. Nevertheless the agents designed in Section 3 will be able to cope with cellular arrays that initially contain spaces.
- The cells on which the agents are placed initially are called source cells. Each agent has stored initially a target location. Source locations may act as targets for other agents, too. The targets are mutually exclusive (each agent has a different target).
- Initially an agent cannot be placed already on its target (message transfers within a cell without movement are not taken into consideration).
- No new messages are inserted into the system until all messages of the current set have reached their targets. This corresponds to a barrier-synchronization between successive sets of messages.

The goal is to find for a given number of k (agents, messages per set, sources/targets) the optimal agents' behavior in order to transfer a message set (averaged over all possible sets) as fast as possible. We are searching for behaviors that are reliably, meaning that the message set transfer can be accomplished successfully for any given initial configuration. From former investigations (Ediger and Hoffmann (2009, 2010)) we have learned that agents with deterministic behavior may run into deadlocks or livelocks. Therefore, we are using here agents with a probabilistic behavior.

2.2 CA Modeling of FSM Controlled Agents

The whole agent system (environment with moving agents) is modeled as a *uniform* CA, e. g. all cells obey to the same local rule. Uniform CA can also model *non-uniform* CA. In our case this is implemented by the use of a *celltype* field as a part of the cell's state. Depending on *celltype*, the relevant subrule is activated. The *celltype* can dynamically be changed by a subrule, e. g. (AGENT $\rightarrow$ EMPTY) for the center cell and (EMPTY $\rightarrow$ AGENT) for the *front cell* (the cell where the agent is moving to). The cell under consideration, also called *center cell* or *own cell* C, is connected to its neighbors within Manhattan distance of 2. The updating scheme is synchronous.

Modeling Moving. Modeling moving agents as CA can be described by two complementary rules (own rule R_C, neighbor's rule R_N) (Fig. 1(a)). If an agent moves from its own location C to one of its neighbor locations N, the own rule deletes the agent and the neighbor's rule copies it. In addition, conflicts have to be detected and resolved in the case that a cell can host only one agent (or a limited number). If more than two agents want to move to the same neighbor, either all agents have to wait or one agent is selected. The neighboring cells (where the agent wants to move to) and the own cell have to perform the

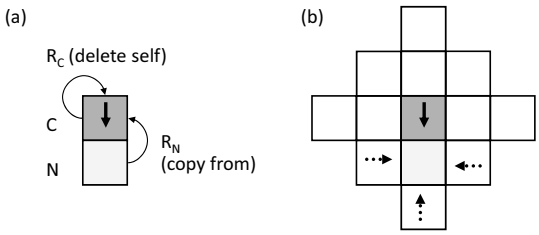


Fig. 1: (a) Modeling the moving of an agent in CA requires a couple of two consistent rules (sender rule R_C deletes agent, receiver rule R_N copies agent). (b) Conflict resolution has to be computed in the sender cell (C) and the receiver cell (N) based on the same information. A neighborhood of distance 2 in the agent's direction is required in order to solve the conflict.

Tab. 1: A state table representing the best found control automaton for the large environments ($N = 256$, see Sec. 4, best of V20). Depending on current control state and inputs, the pairs of *next state*/*output* are given.

CONTROL STATE	INPUT																	
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
0	1/0	5/0	0/0	6/2	3/0	6/3	2/2	3/1	7/1	4/2	1/2	0/0	2/2	3/0	7/1	1/1	3/1	3/1
1	6/2	3/0	2/0	0/2	4/1	7/3	1/2	7/1	4/1	1/0	5/0	2/0	5/2	6/2	1/1	7/2	7/1	5/1
2	2/2	5/2	5/0	4/2	1/1	0/3	6/1	1/1	2/1	3/0	0/0	6/0	3/0	7/1	0/1	6/2	6/1	5/1
3	3/0	0/1	4/3	4/2	1/2	5/1	1/2	0/1	4/1	5/0	2/0	6/0	0/1	2/2	5/1	7/2	3/1	7/1
4	3/0	0/0	6/3	4/2	5/2	2/2	4/1	7/2	6/3	3/0	5/0	2/0	2/2	6/2	3/0	6/2	0/1	4/1
5	3/1	7/0	5/3	7/2	2/2	1/0	7/2	2/1	0/1	4/2	0/0	5/0	4/2	0/2	2/3	6/2	4/1	5/3
6	6/2	6/0	6/3	4/2	7/2	1/0	6/1	2/1	5/3	4/2	2/0	1/0	6/2	0/2	4/3	4/1	5/1	0/1
7	4/2	7/0	7/0	1/2	5/2	4/1	2/2	4/1	3/1	4/2	7/0	5/0	3/2	5/2	1/0	6/1	1/1	3/1

same conflict resolution scheme using the same amount of information consistently. In order to access this information by the neighboring cells as well as by the own cell, in general an extended neighborhood is required (Fig. 1(b)). I. e., C needs to read the state of the front cell N, and in addition the states of the neighbors of N in order to detect a conflict. An empty cell N reads the state of its four neighbors in order to detect an agent that wants to move to it, or to detect a conflict. In Ediger and Hoffmann (2008a) CA rules are given describing the moving more formally.

Modeling Behavior. An agent shall react on certain inputs coming from the environment or from other agents. If an agent behaves according to an internal algorithm that is not trivial, we will call the agent "intelligent". We are using a finite state machine (FSM, Mealy type) defining a control algorithm. The outputs of the control algorithm activate certain actions. The control algorithm together with its actions define the behavior (or the "algorithm") of an agent. The whole agent represents a Moore automaton, which is the type of automaton that is standard in CA.

We represent such control automata by a transition table (Tab. 1), defining the next control state and the control output. It can be implemented easily by a table stored in a read-only memory. The number of FSMs that can be coded in a table with $\#x$ control inputs, $\#s$ states, and $\#y$ control outputs is $(\#s\#x)^{(\#s\#y)}$. As the number of FSMs and the storage capacity is exploding with respect to these parameters, this implementation is of practical use only for a limited complexity. Nevertheless, interesting non-trivial algorithms can be coded with a limited complexity. – Tab. 1 shows an example with $\#x = 18$ inputs, $\#s = 8$ control states and $\#y = 4$ control outputs.

Note that the number of control algorithms which fulfill certain properties (only one representative of

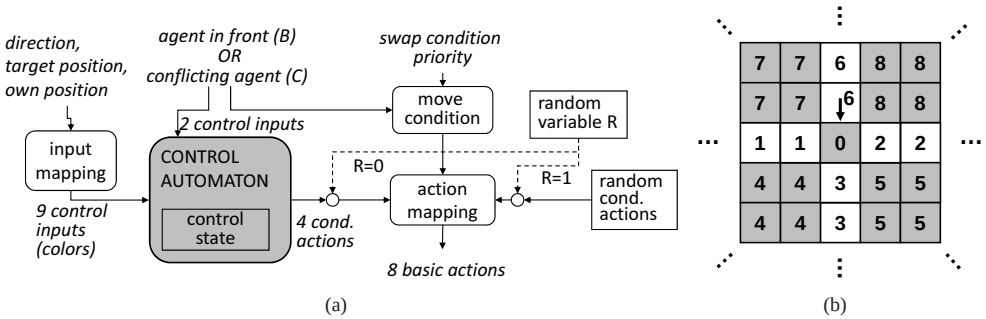


Fig. 2: Control Unit (a). The current direction, and the own and target position of the agent are mapped to 9 control inputs ("colors"). An agent can observe 9 target areas (colors) where the target may be located (b). Two additional control inputs are (B $\vee$ C), altogether 18 control inputs. The control automaton computes 4 control outputs (the conditional actions). The action mapping maps the outputs to the 8 basic actions.

equivalents, state reduced, only one representative of equivalents under permutations of the state/input/output encoding, fully connected, etc.) is much smaller, but still its number is growing exponentially. In Halbach (2008) algorithms are given that allow enumerating only algorithms that fulfill certain properties. In order to keep the complexity of the control automaton under reasonable limit, the inputs are reduced by an *input mapping function* (Fig. 2(a), Fig. 2(b)), and the control outputs are mapped to a larger set of basic actions by an *action mapping function* (using control outputs and other conditions).

Depending on the random variable R, the conditional action is either taken from the control automaton with probability $(1-p)$ or is random with probability p. Thereby the whole behavior gets probabilistic.

The Cell's State and Rule. The state of each cell is structured into (*celltype*, *direction*, *own and target position*, *priority*, *control state*, *random variable R*) (Fig. 3). The celltype is in $\{$AGENT, EMPTY, OBSTACLE$\}$. Space cells are modeled by EMPTY cells in the initial configuration. Each agent has a moving direction (toN, toE, toS, or toW) computed in the current generation for the next generation. Depending on the own position (can be read from the environment, or by the use of a position counter that is updated according to each moving step) and the target position, the agent can compute its shortest path (resp. the most advantageous next direction) to the target. The action taken by the agent depends (i) on the random variable R, (ii) on the inputs, and (iii) the agent's current control state. The eight basic actions are ($k \in (0, 1, ..., 3)$):

- "Move and Turn" mT_k: move forward and simultaneously turn $k \times 90^\circ$ clockwise
- "Stay and Turn" sT_k: stay and turn $k \times 90^\circ$ clockwise

The following shortcuts can be used: $T_0 = N$ (no turn), $T_1 = R$ (turn right), $T_2 = B$ (turn back), $T_3 = L$ (turn left), then the basic actions are *mN*, *mR*, *mB*, *mL*, *sN*, *sR*, *sB*, *sL*.

An agent can perform one out of four *conditional actions*. The conditional action is defined by the control automaton if $R = 0$, or is random (one out of four) if $R = 1$. R is a random variable that is set to one with a probability of p, otherwise to zero. The changing of the random variable is performed in every

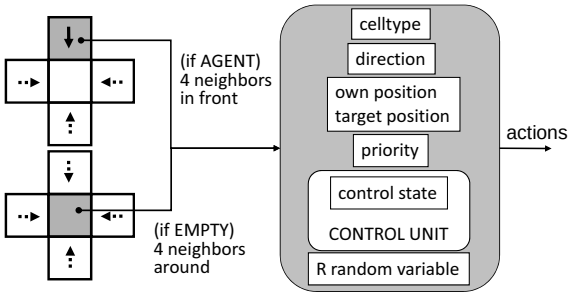


Fig. 3: Structure and neighborhood of an intelligent agent. The actions are determined by a control unit that is embedded in the cell. If the cell type is AGENT, then the 4 neighbors in front are relevant. If the cell type is EMPTY, then the 4 surrounding neighbors are relevant.

generation by a random generator available in every cell. Thereby the behavior of the agents becomes probabilistic. A conditional action depends on the moving condition m: T_k: if m then mT_k else sT_k.

This means that an agent moves forward whenever it can (m = true). Note that in addition to a movement the direction may be changed. The moving condition is given by $m = (\overline{B} \wedge \overline{C}) \vee swap$. B means that there is an agent in front. C means that another agent (the "conflicting agent") is allowed to move to the empty cell in front. The moving condition gets false, if there is a blocking agent in front (B), or if the conflict resolution forbids the agent to move because another agent gets priority (C). The condition *swap* means that the agent in front points back to the own agent; thereby the head-on meeting agents are swapped. Note that swapping is very useful in order to avoid blockings and deadlocks.

In case of a conflict (2-4 agents meet at a "crossing" and point to the same crossing cell), the crossing cell acts as an arbiter. There are $4! = 24$ priority schemes (possible permutations) to resolve the conflict for at most 4 agents. The agent with the highest priority, decided by the crossing, wins and moves to the crossing point. For example, if the crossing priority scheme is (3, 1, 0, 2) and one agent from direction 1 and another one from direction 3 want to move to the crossing point, then the agent coming from direction 3 will win (is predecessor in the list) and moves. The losers wait and perform one of the turning actions *tk*. The 24 schemes are equally distributed over the cell space in the initial configuration (stored in every cell one after the other, repeated cyclically).

Depending on the direction, the own position and the target's position, 9 target areas are distinguished (Fig. 2(b)). The target area 0 contains only the front cell. Four of them are the 4 main directions (lines, main axes), that are defined by the following condition: the target can be reached (be seen) by going straight forward (where required after rotation). Another four areas are given for the sectors in between these lines. Target areas can be interpreted as different colors, which an agent can observe. Our agents can observe all 9 different colors (If necessary, the number of colors could be reduced in order to reduce the inputs for the control automaton to be evolved (Sec. 3).)

The control automaton computes first the preferred conditional action. Then the moving condition (including the swapping option) is checked. Thereby the agent adapts to the given situation in its immediate local neighborhood, because if a desired link is blocked, it will choose an alternative link in the next step, determined by the control automaton and by the priority scheme stored in the front cell. The cell rule can be informally described as follows:

- If celltype is EMPTY, calculate which of the four neighboring cells with cell type AGENT pointing to the own cell has the highest priority. If such an agent exists, copy the control state, direction and the other information and use it in the own control automaton to determine (in the own cell) the turning decision of the agent. Finally change the celltype to AGENT. If the own cell is the target of the agent, then the celltype remains EMPTY (agent is deleted on target).

- If celltype is AGENT, detect whether the movement to the front cell is possible (including the swapping option and taking into account the priority scheme stored in the front cell). If the agent can move, change the cell type to EMPTY.

3 Types of Agents and Investigations

The goal is to find optimal agents to solve the routing problem with $n \times n = N = 16, 56, 256$ cells and agents. These different cases are called *CASE(N)*. Three types of agents will be used: FSM controlled agents (Sec. 3.1), and for comparison (Sec. 3.2) XY-routing agents (XY) and modified XY-routing agents (MXY).

3.1 Method of Determining the FSM Controlled Agents

The probabilistic algorithm is defined by the control algorithm of the finite state machine and the probability p of the randomness parameter. Therefore the genome is composed of an FSM state table (as shown in Tab. 1) and a probability p. In this investigation p can take on discrete values with a precision of 0.1%. A randomized FSM controlled algorithm will be called FSM(p).

Our method to find (near) optimal algorithms consists of the following steps:

1. (*Evolving*) A *training set* of 20 initial configurations was used to evolve a first set of 3,000 algorithms **T3000**. An island model as described in Ediger and Hoffmann (2008a); Ediger et al. (2009) was used. Five islands were used with a population size of 100 genomes (initially randomly generated) and an immigration rate of 2%. Six runs were performed, resulting in $5 \times 100 \times 6 = 3000$ algorithms. The fitness value was evaluated for each configuration by using only one simulation of the CA. It is equal to the number of steps (1/"speed"), that the agents need to complete the message set transfer (averaged over the 20 configurations).

   New automata were constructed during the evolution process using a uniform crossover with two parent automata like in Ediger and Hoffmann (2008a); Ediger et al. (2009). Thereby the next state of a control state and its associated output (conditional action) can be taken from either one of the parents. The probability (stored in the genome) of the offspring is the average of the probabilities taken from the parents. In addition, each gene (next state, output, probability) is changed with a mutation rate of 0.9%.

2. (*Fitness Correction*) The fitness values of the evolved algorithms T3000 are not precise because they are evaluated by one simulation only. Different simulations lead to slightly different fitness values because the CA rule is probabilistic. Therefore the 3,000 algorithms were simulated again, each 1,000 times. Thereby the confidence into the fitness values was increased significantly. Then the algorithms T3000 were ranked and the top 20 were selected for further processing, let us call them **T20**.

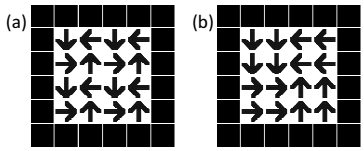


Fig. 4: Manually designed initial configurations for $N = 16$ that lead to livelocks or deadlocks if the control algorithm is unrandomized and equal for all agents. In case (a) the target of each agent is placed in its front cell. In case (b) the target is placed straight ahead of the agent, two cells in front. The target positions are not depicted here.

3. (*Ranking*) We want to produce algorithms that are successful and fast on any initial configuration. Until now the algorithms were optimized for the training set only. Therefore another, larger set, called *ranking set* containing 1,000 configurations was used. It consists of 998 random initial configurations and 2 manually designed configurations shown in Fig. 4. We have designed these two special configurations in such a way that livelocks or deadlocks will occur if unrandomized algorithms (deterministic FSMs, deterministic XY-routing (see below)) were used. The algorithms T20 were simulated 10,000 times (for $N = 16, 64$ cells), and only 1,000 times (for $N = 256$) in order to limit the computation time. Then they were ranked yielding the set of **R20** algorithms.

4. (*Variation*) As randomness p of an algorithm was evolved for the training set only and only simulating once, its confidence is low. We are searching for the optimal randomness for the ranking set. Therefore the randomness was varied using discrete values with a precision of 0.1%; lower and higher values were used in an iterative way in order to find the optimal value. The simulations were carried out 10,000 times (for $N = 16, 64$) resp. 1,000 times (for $N = 256$) on the 1,000 initial configurations of the ranking set for different values of p. The set of algorithms that results from variation is called **V20**.

3.2 Randomized XY-Routing Agents and Modified XY-Routing Agents

For comparison, XY-routing agents (XY)[(i)] and modified XY-routing agents (MXY) were defined and simulated. These agents can see the same colors as the FSM controlled agents, and they can perform the same basic actions. The state of an XY or an MXY is (*direction*, *own position*, *target position*).

An XY first reduces the difference of the x-coordinates between the current and target position until it gets zero, then it reduces the y-difference.

An MXY computes in each step the optimal moving direction from the cell in front towards the target on the shortest path. If there are two equivalent directions, then it takes an arbitrary choice. If the front cell is free, then the agent will move and turn towards the target. If the front cell is blocked and there are alternative directions, then the agent takes an arbitrary choice.

For certain configurations, it turned out that that the XYs and MXYs formed clusters (deadlocks and livelocks) from which they could not escape. Therefore, the capabilities of the XYs and the MXYs were enhanced by adding randomness in a similar way as it was done for the FSM agents. An XY(p) is an XY that turns to a random direction with a randomness of p, and behaves as an XY for $(1 - p)$. The same holds for MXY(p). The optimal randomness was searched in the same way as described above in step 4 (variation) (Sec. 3.1).

(i) The "normal" XY-routing procedure uses four message buffers per cell, whereas here only one buffer for the agent is provided.

Tab. 2: Fitness and randomness values of the Top1 algorithms of R20 and V20, the MXY(p) algorithm and the XY(p) algorithm for each *CASE(N)* on the Ranking Set. Note that the best algorithm of R20 is not necessarily the best algorithm of V20.

	N = 16		N = 64		N = 256	
	fitness	p	fitness	p	fitness	p
Top1 of V20	**14.8**	**1.2%**	**36.7**	**1.3%**	**84.0**	**1.8%**
Top1 of R20	15.4	4.5%	37.6	0.6%	86.4	4.4%
MXY(p)	**15.8**	**6.5%**	**41.1**	**9.1%**	**97.8**	**11.9%**
XY(p)	17.9	8.4%	50.2	14.7%	122.7	18.5%

Tab. 3: Average, minimum and maximum of the randomness p of the FSM algorithms in R20 and V20.

	FSMs(p) of V20, optimal p			FSMs(p) of R20, evolved p		
	N = 16	N = 64	N = 256	N = 16	N = 64	N = 256
Average randomness	3.1%	1.89%	2.19%	3.1%	4.78%	5.69%
Minimum randomness	0.9%	0.8%	1.2%	0.1%	0.6%	2.9%
Maximum randomness	5.6%	3.1%	3.2%	6.1%	7.6%	7.5%

4 Performance of the Agents

4.1 Performance of the Randomized Algorithms

The fitness (1/"speed") of the found algorithms is shown in Tab. 2. For each *CASE(N)* the fitness is ordered with this precedence: (1) Top1-FSM(p) of V20, (2) Top1-FSM(p) of R20, (3) MXY(p), (4) XY(p). Thus the evolved FSM controlled agents (after randomness variation) are 6%/11%/14% faster than the MXY(p) agents, for the cases $N = 16/64/256$. By variation of the randomness p (of FSM(p) in R20) the performance was improved by 2-4% (result of variation is V20).

The largest possible distance between an agent and its target in an initial configuration is $(n-1)+(n-1)$. Supposing, that such an agent is facing a border at the beginning, there is one additional step necessary to turn in the right direction. In such a worst case situation the message (and thus the message set) could be transported in $2n-1$ steps (assuming a perfect algorithm and no conflicts), which equals 7, 15, or 31 for the different cases. In *CASE(16)* 2.1 times this limit is needed by the best algorithm (V20), in *CASE(64)* the factor is 2.4 and in *CASE(256)* the factor is 2.7. This means that the agents move slower to their targets, when there are more agents, because the amount of conflicts per agent per time step is increasing.

4.2 The Optimal Randomness

It was observed that, the better the algorithm performs, the lower is the optimal randomness. The optimal randomness increases with N for the MXY(p), the XY(p) and the Top1 V20 algorithms (Tab. 2). After the ranking step, i. e., after evolving, the average randomness of the R20 algorithms increases with N, too (Tab. 3). For the average randomness of the V20 algorithms and the randomness of the Top1 R20 algorithm, this does not hold for *CASE(16)*. A higher randomness value is needed to efficiently resolve the higher amount of conflicts in configurations with more agents.

Comparing the randomness values after evolving and ranking with the values after variation, it can be

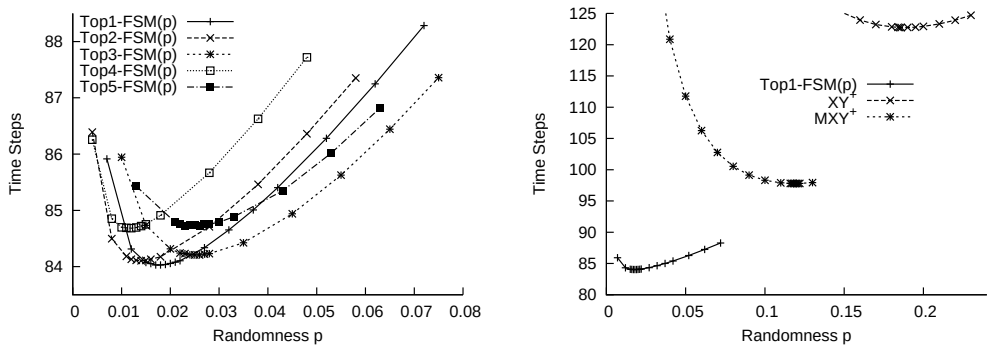


Fig. 5: Curves showing the fitness values for *CASE(256)* depending on the randomness. The set of curves (a) shows the top 5 FSM(p) algorithms of V20. The curves (b) show the fitness values of the XY(p), the MXY(p) and the Top1-FSM(p) algorithm of V20. For *CASE(16)* and *CASE(64)* (not depicted here), the values are different, but the tendency is the same.

observed that a rather high randomness was evolved and later corrected downwards by variation. After variation, also the range of randomness in which the 20 algorithms are placed becomes smaller (Tab. 3).

By performing the variation step, a fitness curve depending on the randomness can be developed. These curves of the best randomized FSMs have a similar progression (Fig. 5(a)). Furthermore, the optimal randomness p of the FSM based evolved algorithms is much lower than the optimal randomness of the MXY(p) and the XY(p) (Tab. 2, Fig. 5(b)). This is true for all tested cases.

4.3 Adding Spaces between the Communication Nodes

An additional test was carried out in order to verify former results saying that 1-2 spaces between each agent will improve the speed of the agents. We investigated initial configurations with $k = 256$ agents and a grid size of $(n + i) \times (n + i)$. This results in $2in + i^2$ spaces (empty cells, buffers). i was varied between 1 and 15. For each of the different cases i, a ranking set of 1,000 configurations was generated, placing the spaces randomly. The Top1-FSM(p) algorithm of V20 of *CASE(256)* was simulated 1,000 times on each ranking set. It turned out that for $i = 11$ it performs best: 69.3 time steps on average, which is an improvement of 17.5% against the configurations with $i = 0$ (84 time steps). For $i = 11$, there are 420 spaces. Thus the best ratio of agents to spaces is 1.64.

5 Conclusion

Agents modeled within the CA paradigm were developed that can efficiently solve the routing problem. Three types of agents were defined: (1) FSM controlled agents (with an embedded finite state machine evolved by a genetic algorithm), (2) XY-routing agents, and (3) modified XY-routing agents (MXY). To each of them a randomness of p was added: With the probability p they choose an arbitrary moving action in order to avoid deadlocks and livelocks. The FSM agents were first evolved on a small training set of initial configurations, then their fitness values were corrected using more simulations, then they were

ranked on a large training set, and finally the evolved randomness parameter p was varied in order to find its optimal value. It turned out that the FSM(p) agents are the fastest, the MXY(p) agents are slower, and the XY(p) agents are the slowest. Furthermore, if an agent performs better then it uses a lower randomness (it is "more deterministic"). An additional test showed that the speed of the FSM(p) agents can further be improved by introducing additional spaces between the communication nodes. Using $1.64 \times k$ space cells for $k = 256$ agents, the speed could be improved by 17.5%. – Future questions are: Is a regular distribution of the space cells better than a random distribution? Is it better to use different node locations for sending and receiving messages? How fault tolerant is the agent system, replacing some of the spaces by obstacles?

References

C. Busch, M. Herlihy, and R. Wattenhofer. Routing without Flow Control. In *SPAA*, pages 11–20, New York, 2001. ACM SIGACT, ACM SIGARCH, ACM Press.

G. D. Caro and M. Dorigo. AntNet: A Mobile Agents Approach to Adaptive Routing. Technical Report 97-12, IRIDIA, Université Libre de Bruxelles, Belgium, Nov. 09 1997.

A. V. de Mello, L. C. Ost, F. G. Moraes, and N. L. V. Calazans. Evaluation of Routing Algorithms on Mesh Based NoCs. Technical Report 040, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, 2004.

S. Dhillon and P. V. Mieghem. Performance Analysis of the AntNet Algorithm. *Computer Networks*, 51 (8):2104 – 2125, 2007.

P. Ediger and R. Hoffmann. Optimizing the Creature's Rule for All-to-All Communication. In *EPSRC Workshop Automata-2008. Theory and Applications of Cellular Automata, Bristol, UK*, pages 398–410, 2008a.

P. Ediger and R. Hoffmann. Improving the Behavior of Creatures by Time-Shuffling. In H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki, and S. Bandini, editors, *ACRI*, volume 5191 of *LNCS*, pages 345–353. Springer, 2008b.

P. Ediger and R. Hoffmann. CA Models for Target Searching Agents. In P. P. B. de Oliveira and J. Kari, editors, *Proceedings of Automata 2009: 15th International Workshop on Cellular Automata and Discrete Complex Systems, São José dos Campos, Brazil*, pages 41–54, 2009.

P. Ediger and R. Hoffmann. Routing Based on Evolved Agents. In *23rd PARS Workshop on Parallel Systems and Algorithms, Hannover, Germany*, pages 45–53, 2010.

P. Ediger, R. Hoffmann, and M. Halbach. Evolving 6-state Automata for Optimal Behaviors of Creatures Compared to Exhaustive Search. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *EUROCAST*, volume 5717 of *LNCS*, pages 689–696. Springer, 2009.

M. Goldenberg, A. Kovarsky, X. Wu, and J. Schaeffer. Multiple Agents Moving Target Search. In G. Gottlob and T. Walsh, editors, *IJCIA*, pages 1536–1538. Morgan Kaufmann, 2003.

M. Halbach. *Algorithmen und Hardwarearchitekturen zur optimierten Aufzählung von Automaten und deren Einsatz bei der Simulation künstlicher Kreaturen*. PhD thesis, Technische Universität Darmstadt, 2008.

M. Halbach, R. Hoffmann, and L. Both. Optimal 6-State Algorithms for the Behavior of Several Moving Creatures. In S. El Yacoubi, B. Chopard, and S. Bandini, editors, *ACRI*, volume 4173 of *LNCS*, pages 571–581. Springer, 2006.

S. Koenig, M. Likhachev, and X. Sun. Speeding up Moving-Target Search. In E. H. Durfee et al., editor, *AAMAS, Honolulu, Hawaii, USA*, pages 1144–1151. IFAAMAS, 2007.

M. Komann, A. Mainka, and D. Fey. Comparison of Evolving Uniform, Non-uniform Cellular Automaton, and Genetic Programming for Centroid Detection with Hardware Agents. In V. E. Malyshkin, editor, *PaCT*, volume 4671 of *LNCS*, pages 432–441. Springer, 2007.

M. Komann, P. Ediger, D. Fey, and R. Hoffmann. On the Effectivity of Genetic Programming Compared to the Time-Consuming Full Search of Optimal 6-State Automata. In L. Vanneschi, S. Gustafson, and M. Ebner, editors, *EuroGP 2009*, LNCS, Tübingen, Apr.15-17 2009. Springer.

R. E. Korf. Real-Time Heuristic Search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

P. K. K. Loh and E. C. Prakash. Performance Simulations of Moving Target Search Algorithms. *Int. J. Comput. Games Technol.*, 2009:1–6, 2009.

A. Schadschneider. Conflicts and Friction in Pedestrian Dynamics. In H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki, and S. Bandini, editors, *ACRI*, volume 5191 of *LNCS*, pages 559–562. Springer, 2008.

A. Schadschneider and M. Schreckenberg. Cellular Automaton Models and Traffic Flow. *J. Phys A*, 26: L679, 1993.

M. Sipper. *Evolution of Parallel Cellular Machines, The Cellular Programming Approach*, volume 1194 of *LNCS*. Springer, 1997.

M. Sipper and M. Tomassini. Computation in Artificially Evolved, Non-uniform Cellular Automata. *Theor. Comput. Sci.*, 217(1):81–98, 1999.

A. Spicher, N. Fatès, and O. Simonin. From Reactive Multi-Agents Models to Cellular Automata - Illustration on a Diffusion-Limited Aggregation Model. In Joaquim Filipe et al., editor, *ICAART*, pages 422–429. INSTICC Press, 2009. ISBN 978-989-8111-66-1.

Y. Tavakoli, H. H. S. Javadi, and S. Adabi. A Cellular Automata Based Algorithm for Path Planning in Multi-Agent Systems with A Common Goal. *IJCSNS, International Journal of Computer Science and Network Security*, 8(7):119–123, 2008.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 221–230

Randomness solves the density classification problem with an arbitrary precision

Nazim Fatès

[1] *INRIA Nancy – Grand Est& LORIA, campus scientifique, BP 239, F-54506 Vandœuvre lès Nancy, France*

The density classification problem consists in making a population of cells decide, by using only local rules, whether an initial random configuration contains more 0s or 1s. This problem is known for having no exact solution in the case of deterministic one-dimensional cellular automata. We propose a probabilistic cellular automaton that solves the problem with an arbitrary precision. The precision of the classification can be increased with an appropriate tuning of the CA but comes at a cost of an increased average number of steps times to converge.

Keywords: density classification problem ; probabilistic CA ; discrete dynamical systems

1 Introduction

The density classification problem is one of the most studied "inverse" problems in the field of cellular automata. Its interest stems from the paradox that it requires a cellular automaton, or more generally a discrete dynamical system, to compute a trivial task: to decide whether an initial binary string contains more 0s or more 1s. In its most "classical" formulation, the cells are arranged in a ring and each cell can only read its own state and the states of the neighbouring cells. The challenge is to design a local behaviour of the cells that would drive the system to converge to a uniform fixed point, consisting of all 1s if the initial configuration contained more 1s and all 0s otherwise. In short, the cellular automaton should decide whether the initial density of 1s was greater or lower than 1/2.

The impossibility to centralise the information is the main difficulty of this problem. All the computations are local, that is, they are restricted in space and time by the very nature of a two-state cellular automaton. To reach a global consensus which consists of a uniform state, the cells need to compute the state that is most present in their neighbourhood and then to propagate this information to the other cells. However, this can be achieved only with a trade-off between two contradictory objectives: to decide locally which state is most present and, as the cells have no memory, to *simultaneously* propagate this information to the other cells.

The problem has attracted a sustained attention since its early formulations [5]. Studies were mainly conducted an experimental basis, for instance by using genetic algorithms to find good rules (see *e.g.* [7] and references therein). On the analytical side, one of the most surprising discoveries was a negative statement: there exists no perfect (deterministic) density classifier that uses only two states [6]. This draft paper presents a kind of "counterpart" to this negative theorem: even though it is impossible to find a

perfect solution, the density classification problem may be solved with an arbitrary precision, *i.e.*, with a probability of success arbitrarily close to 1.

Our main idea is to use randomness to solve the dilemma between the local majority decisions and the propagation of the most frequent state on a global scale. We follow the path opened by H. Fukś who proposed a probabilistic CA which classifies density by a purely "diffusive" mechanism. In our construction, the trade-off between local majority computations vs. large-scale diffusion is achieved by tuning a single parameter. This parameter corresponds to a weight between two well-known deterministic rules, namely the majority rule and the "traffic" rule. We show that the probability of making a good classification approaches 1 as the value this parameter is set closer to 0. The drawback of such a gain of precision is an increase in the average time it takes to converge to a uniform state.

2 Formalisation of the problem

In this section, we define the Elementary Cellular Automata and their probabilistic counterpart.

2.1 Basic notations

Let $\mathcal{L} = \mathbb{Z}/n\mathbb{Z}$ be the set of n cells arranged in a ring. We restrict our study to the binary case, the set of states is $\{0,1\}$. A *configuration* is a string $x \in \{0,1\}^{\mathcal{L}}$ that associates to each cell a state. The set of all configurations of size n is denoted by $\{0,1\}^{\mathcal{L}}$.

An *Elementary Cellular Automaton* (ECA) is a one dimensional binary CA with nearest neighbour topology, defined by its *local transition rule*, a function $f : \{0,1\}^3 \to \{0,1\}$ that specifies how to update a cell using only nearest-neighbour information. For a given ring size n, the *global transition rule* $F : \{0,1\}^{\mathcal{L}} \to \{0,1\}^{\mathcal{L}}$ associated to f is the function that maps a configuration x^t to a configuration x^{t+1} such that:

$$\forall i \in \mathcal{L},\ x_i^{t+1} = f(x_{i-1}^t, x_i^t, x_{i+1}^t)$$

A *Probabilistic Elementary Cellular Automaton* (**P-ECA**) is also defined by a *local probabilistic transition rule* f but the next state a cell is known only with a given probability. In the binary case, we define $f : \{0,1\}^3 \to [0,1]$ where $f(x,y,z)$ is probability that the cell updates to state 1 given that its neighbourhood has the state (x,y,z).

We define the *global transition rule* F associated to a **P-ECA** f as the probabilistic function that assigns to each random configuration x^t a random configuration x^{t+1} such that:

$$\forall i \in \mathcal{L},\ x_i^{t+1} = \mathcal{B}_i^t\big\{\, f(x_{i-1}^t, x_i^t, x_{i+1}^t)\,\big\}$$

where x_i^{t+1} denotes the random variable that is given by observing the state of cell i and $\mathcal{B}_i^t(p)$ are i.i.d. Bernoulli random variables, *i.e.*, random variables that equal to 1 with probability p and to 0 with probability $1-p$.

2.2 Density Classifiers

We say that a configuration x is a *fixed point* for the global function F if we have $F(x) = x$ with probability 1. We say that a global function F is a *(density) classifier* if $0^{\mathcal{L}}$ and $1^{\mathcal{L}}$ are its two only fixed points.

Tab. 1: Table of the 8 active transitions and their associated letterd that define the notation by transitions of the 256 ECAs.

A	B	C	D	E	F	G	H
000 1	001 1	100 1	101 1	010 0	011 0	110 0	111 0

For a classifier $\mathcal{C}$, we define the probabilistic event that $\mathcal{C}$ *correctly classifies* a configuration x as the probability that there exists a finite time T such that: $\mathcal{C}^T(x) = 1^{\mathcal{L}}$ if $d(x) > 1/2$ and $\mathcal{C}^T(x) = 0^{\mathcal{L}}$ if $d(x) < 1/2$.

To evaluate the "quality" of a classifier requires to introduce a quantitative measures. We use here the "uniform density quality" UDQ, defined as the limit for the ring size growing to infinity of the probability of good classification if the initial strings are constructed as Bernoulli strings, *i.e.*, we choose d uniformly in $[0, 1]$ and construct a random configuration where each cell has a probability d to be in state 1.

2.3 Structure of the ***P-ECA*** space

Obviously, the classical deterministic ECA are particular **P-ECA** with a local rule that takes its values in $\{0, 1\}$. The space of **P-ECA** can be described as an eight-dimensional hypercube with the ECA in its corners. This can be perceived intuitively if we see **P-ECA** rules as points of the hypercube, to which we apply the operations of addition and multiplication. More formally, taking p **P-ECA** $F_1, \ldots, F_k$ and $w_1, \ldots, w_k$ real numbers in $[0, 1]$ such that $\sum_{i=0}^{k} w_i = 1$, the weighted average of the **P-ECA** (F_i) with weights w_i is the **P-ECA** g such that:

$$\forall x, y, z \in \{0, 1\},\ g(x, y, z) = \sum_{i=0}^{k} p_i.f(x, y, z)$$

As a consequence, one may choose any combination of 8 **P-ECA**that form a basis as vector coordinates of the 8-dimensional hypercube. The most intuitive basis is the 8 ECA that have only one transition that leads to 1: the weights of this combination correspond to the values $f(x, y, z)$.

Equally, one may express a **P-ECA** as a weighted average of the 8 (deterministic) ECA that have only one *active* transition, *i.e.*, only one change of state in their transition table. Such ECA are labelled A, B, ..., H according to the notation introduced in [2] and summed up in table Tab. 1. Formally, for every **P-ECA** f, there exists a 8-tuple $(p_{\mathrm{A}}, p_{\mathrm{B}}, \ldots, p_{\mathrm{H}})$ such that:

$$f = p_{\mathrm{A}}.A + p_{\mathrm{B}}.B + \cdots + p_{\mathrm{H}}.H$$

We denote this relationship by $f = [p_{\mathrm{A}}, p_{\mathrm{B}}, \ldots, p_{\mathrm{H}}]_T$, where the subscript T stands for (active) "transitions".

This basis has the same advantages as for the deterministic case (see ref [3]). In particular one can notice that the quantities $p_{\mathrm{A}}, p_{\mathrm{B}}, p_{\mathrm{C}}, p_{\mathrm{D}}$ and $p_{\mathrm{E}}, p_{\mathrm{F}}, p_{\mathrm{G}}, p_{\mathrm{H}}$ concern the cells with state 0 and 1, respectively. The group of symmetries of a rule can easily be obtained: the left-right symmetry permutes p_{B} and p_{C}, and p_{F} and p_{G}, whereas the 0-1 symmetry permutes p_{A} and p_{H}, p_{B} and p_{G}, etc.

3 Fukś density classifier

Let us first consider the probabilistic density classifier proposed by Fukś [4]. For $p \in]0, 1/2]$, the local rule **C1** is defined with the following transitions:

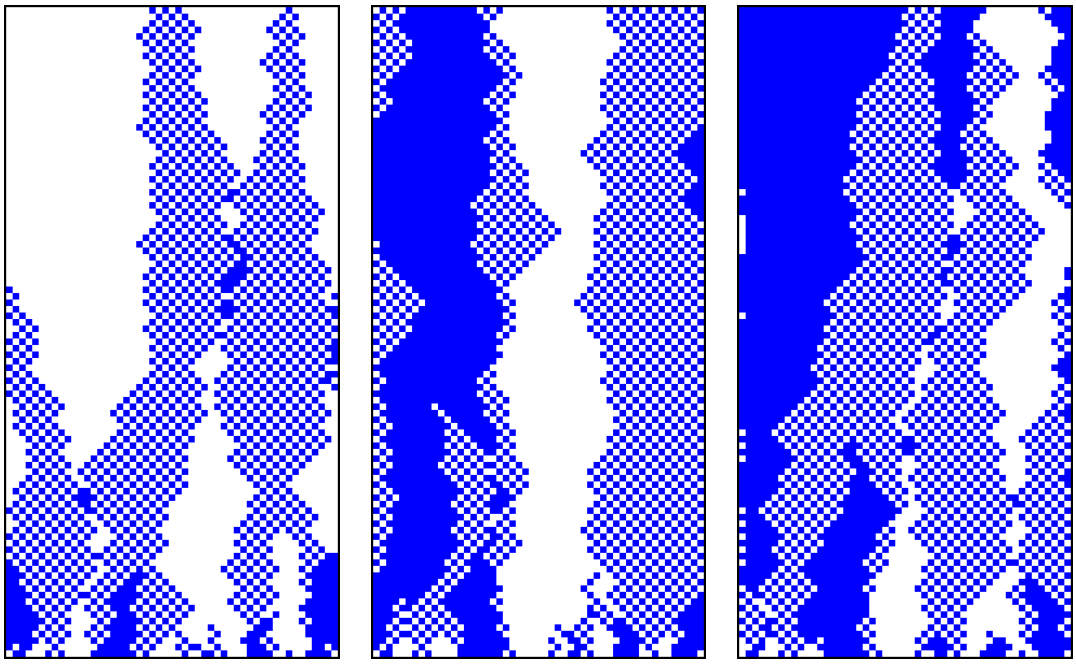


Fig. 1: Three evolutions of Fukś classifier **C1** with $n = 51$, $p = 1/2$, and same initial of density ~ 0.4. Time goes from bottom to top ; white cells are 0-cells and blue cells are 1-cells. (left) evolution will most probably end with a good classification ($0^{\mathcal{L}}$); (middle) equal probabilities of good classification; (right) evolution will probably end with a bad classification ($1^{\mathcal{L}}$).

$\mathcal{N}$	000	001	010	011	100	101	110	111
$p(1\,\|\,\mathcal{N})$	0	p	$1-2p$	$1-p$	p	$2p$	$1-p$	1

This rule is a density classifier as $0^{\mathcal{L}}$ and $1^{\mathcal{L}}$ are its only fixed points. With notations introduced above, we write:

$$\begin{aligned} \mathbf{C1} &= [0, p, p, 2p, 2p, p, p, 0] \\ &= p.BDEG + p.CDEF \end{aligned}$$

where $BDEG = 170$ and $CDEF = 240$ are the left and right shift respectively. This means that Fuks' rule can be interpreted as applying, for each cell independently: (a) q left shift with probability p, (b) a right shift with probability p, and (c) staying in the same state with probability $1-2p$ (see Fig. 1). We also note that this rule is invariant under both the left-right and the 0-1 symmetries ; indeed, we have: $p_{\mathrm{B}} = p_{\mathrm{C}} = p_{\mathrm{F}} = p_{\mathrm{G}}$, $p_{\mathrm{A}} = p_{\mathrm{H}}$ and $p_{\mathrm{D}} = p_{\mathrm{E}}$.

Theorem 1 *For every $x \in \{0,1\}^{\mathcal{L}}$, the probability of good classification of x is equal to* $\max\{d(x), 1-d(x)\}$*, where $d(x)$ is the density of x.*

This property was observed experimentally with simulations explained partially by combinatorial arguments [4]. We now propose a proof that uses the analytical tools developed for asynchronous ECAs [3] and completes the results established by Fukś.

The proof stands on the following lemma, see [3].

Lemma 1 *If (X_t) is a process that takes its values in $\{0,\dots,n\}$, such that:*

- *(X_t) is a martingale on $\{0,\dots,n\}$,* i.e.*, $\mathbb{E}\{\, X_{t+1} \,|, \mathcal{F}_t \,\} = X_t$,*
- *(X_t) is a Markov process with 0 and n as the two only absorbing states,*

then the probability of absorption by state n is equal to X_0/n, the probability of absorption by state 0 is equal to $(n - X_0)/n$.

Proof Proof of Theorem 1: We denoted by $|x|_{\mathrm{P}}$ the number of occurrences of a pattern P in x. Let us now simply take $X_t = |x^t|_1$ and show that Lemma 1 applies to X_t. For sake of simplicity, we write $a(x) = |x|_{000}$, $b(x) = |x|_{001}$, …, $h(x) = |x|_{111}$ (see Tab. 1) and drop the argument x when there is no ambiguity.where $|x|_{\mathrm{P}}$ denotes the number of occurrences of p in x. The following equalities hold [3]: $b + d = e + f$; $c + d = e + g$; $b = c = f = g$.

We thus have:

$$\begin{aligned} \mathbb{E}\{\, X_{t+1} - X_t \,\} &= p.b + p.c + 2p.d - 2p.e - p.f - p.g \\ &= p.(b + d - e - f) + p.(c + d - e - g) \\ &= 0 \end{aligned}$$

On the other hand, X_0 and X_n are the only two absorbing states of the Markov chain, so $X_t \in \{1, n-1\}$ implies that x^t is not a fixed point

Let T be the random variable equal to the time where the system reaches an absorbing state, then T is a stopping time and we can write:

$$E[X_T] = 0.\Pr[X_T = 0] + n.\Pr[X_T = n]$$

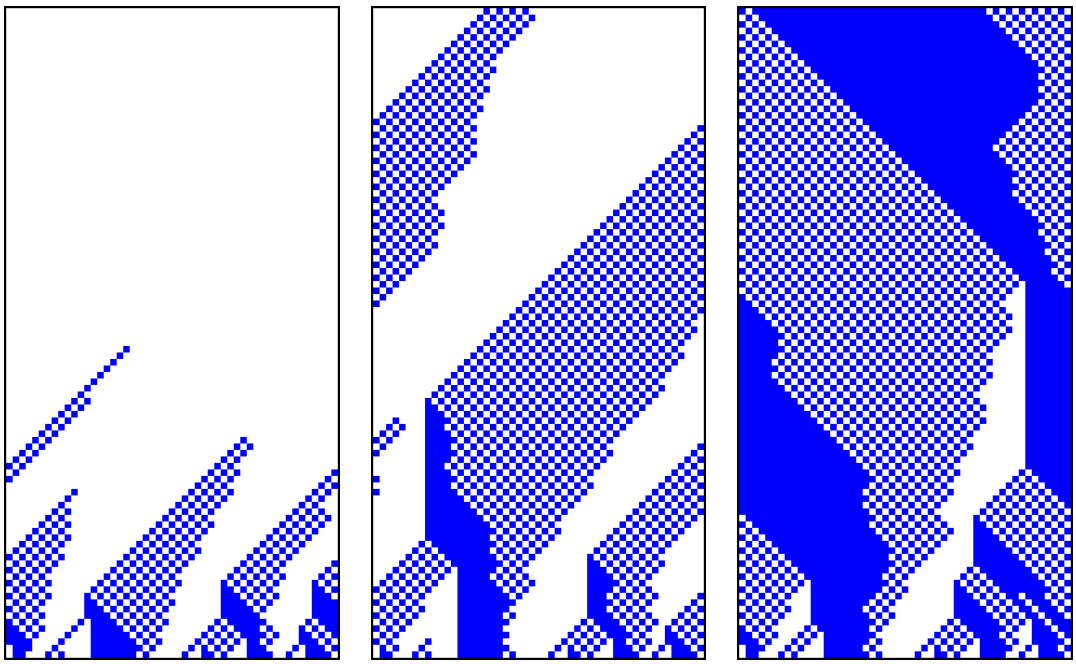


Fig. 2: Three evolutions of **C2** with $n = 51$ and $\epsilon = 1/4$ and for the same initial condition of density $25/51 \sim 0.49$. The two first evolutions give a good classification (all 0) in less or approximately 100 time steps; the third evolution is not finished at time $t = 100$, but it will evolve toward a bad classification.

and

$$E[X_T] = X_0.$$

We find that the probability that the chain stops on $X_t = n$, that is on the fixed point $1^{\mathcal{L}}$, is equal to X_0/n, *i.e.*, it is equal to the initial density. □

From this, we derive that the probability of good classification of any configuration x is equal to $\max\{d(x), 1 - d(x)\}$. The uniform density quality of **C1** is thus equal to $3/4$ (obtained y a simple integration).

It is possible to estimate the convergence time of this classifier by using the same techniques as for the asynchronous ECAs [3]. Indeed, by noting that the Markov chains that describe **C1** and the shift are similar, this time should scales as n^2/p. However, a precise proof of this statement yet remains to be done.

4 Our proposition

For $\epsilon \in [0, 1]$, let us consider the following P-ECA:

$\mathcal{N}$	000	001	010	011	100	101	110	111
$p(1 \mid \mathcal{N})$	0	0	0	1	$1-\epsilon$	1	ϵ	1

Intuitively, this probabilistic rule can be considred as a mixing of two elementary rules.

- For $\epsilon = 0$ we have ECA 184, which is a well-known rule, often called the "traffic" rule. This rule is number conserving, *i.e.*, the number of 1s is conserved as the system evolves (see *e.g.*, [1]). Observing the evolution of the rule, we see that a 1 with a 0 at its right moves to right while a 0 with a 1 at its left is moved to the left. So all happens if the 1s were cars that tried to go to the right, with possible traffic jams. These jams resorb by going in the inverse directions of the cars (when possible).

- For $\epsilon = 1$, we have ECA 232 which is the "majority rule". This rule acts by selecting the state that is most present in the neighbourhood of the cell.

With the notations introduced above, we have:

$$\begin{aligned} \textbf{C2} &= [0, 0, 1-\epsilon, 1, 1, 0, 1-\epsilon, 0]_T \\ &= \epsilon.DE + (1-\epsilon).CDEG \end{aligned}$$

For $\epsilon \in]0, 1[$, the effect of the rule is the same as if, for each cell and each time, we would apply ECA 232=DE with probability ϵ and ECA 184=CDEG with probability $1 - \epsilon$ (see Fig. 2). This combination generates a surprising property: although the system is stochastic, there exists an infinity of configurations which can be classified with no error.

Definition 1 *For $q \in \{0, 1\}$, a configuration x is a q-*archipelago *if all the cells in state q are isolated,* i.e.*, if x does not contain two adjacent cells in state q.*

Theorem 2 *For a given odd ring size n, for each $p \in [0, 1($, there exists an ϵ such that for each configuration $x \in \{0, 1\}^{\mathcal{L}}$, the probability of good classification of x by **C2** is greater than $1 - p$.*

The theorem stands on the two lemmas that follow.

Lemma 2 *For a given odd ring size n, an archipelago is well-classified with probability 1.*

Proof (Sketch): The proof is simple and relies on two observations. Witout loss of generality, let us assume that x is a 1-archipelago, we then have $d(x) < 1/2$. First, the successor of an archipelago is an archipelago (effect of ECA 184). Second, each isolated 1s can disapear with probability ϵ. As a result, all the 1 will eventually disappear and the system will attain the fixed point $0^{\mathcal{L}}$, which corresponds to a good classification.

□

Lemma 3 *For a given odd ring size n, for every $p \in [0, 1($, there exists a setting ϵ of the classifier such that every configuration configuration $x \in \{0, 1\}^{\mathcal{L}}$ has a probability greater than p to evolve to a 1-archipelago (to a 0-archipelago) if $d(x) > 1/2$ (if $d(x) < 1/2$, respectively).*

Proof (Sketch): The proof relies on the well-known properity of rule 184 to evolve to an archipelago in at most $n/2$ steps.

For a given p and given n, without loss of a generality, let us consider a configuration x such that $d(x) < 1/2$. Let us consider the probability that the rule does *not* behave like rule 184. This can happen only on the "b" and "f" cells, *i.e.*, on the cells whose neighbourhood are 100 and 110, with probability ϵ of each such cell. As we have $b = f = g = c$, we can write $b + f \leq n/2$. At each time step, the probability

p_{diff} that the evolution of **C2** and rule 184 differ on one step is thus majorated by: $p_{\text{diff}} \leq \epsilon^{n/2}$. For T steps, it is is majorated by: $p_{\text{diff}} \leq 1-(1-\epsilon^{n/2})^T$.

For a given T, by using the equality above, we find that it is sufficient to take:

$$\epsilon < \left(1-(1-p)^{1/T}\right)^{2/n}$$

to guarantee that the probability of occurence of a difference during T steps is less than p.

This inequality is a only gross majoration but it shows that, by taking ϵ small enough, the probability that a configuration x with $d(x) < 1/2$ evolves to a 0-archipelago can be made arbritrarily small. □

Combining the two lemmas to prove the theorem is straightforward: for ϵ small enough, the system evolves to an archipelago which has the same density as the initial condition (prop. of rule 184). It is then well-classified as it will progressively "drift" towards the approriate fixed point. Note that in most cases, there is no need that these two phases occur sequentially. This indicates that the bounds given above can be largely improved.

The estimation of the time of convergence of this classifier is more complex than for Fukś classifier, as it is not easy to see which quantities are conserved during the evolution of the system. However, by noting that the probability of changing the density of a configuration vanishes as ϵ vanishes, it is clear that the convergence time of the classifier diverges as ϵ tends to 1 and as the quality of classification tends to 1.

5 Discussion

This work-in-progress report presented preliminary results on how a probabilistic cellular automaton can be used to solve the density classification problem with an arbitrary precision. Our proposition consisted in finding a "blend" between two rules that have appropriate properties to solve this particular problem. It is interesting to determine how to "blend" other rules, especially rules with a larger radius or on higher dimension grids. Far from solving the problem, the existence of such a rule suggests that the quality of classification cannot be taken as the unique criterion for evaluating the classifiers. Instead, it is some trade-off between quality and time to give an answer that has to be looked for.

The first informal experiments conducted showed us a good quality of classification, with an answer given within a short simulation time, at least for small-size configurations. For instance, for a ring size of $n = 51$ and $\epsilon = 0.1$, most initial configurations are correctly classified provided their density is not $25/51$ or $26/51$, *i.e.*, if they are not "too close" from density $1/2$. It is now necessary to estimate this data with large statistical measures. Determining these measures analytically is also a challenging problem. We believe that the techniques used to estimate Fukś classifier could be adapted to our classifier, even though such an adaptation does not seem straightforward.

Most of the results so far have been given by using the uniform density quality. As far as we know, it is an open problem to find a classifier that would have has a uniform *configuration* quality which differs from 1/2. In other words, is there a classifier which would give a non-random answer when the configurations are uniformly chosen with large-size grids?

Acknowledgements

The author wishes to express his gratitude to H. Fukś for the stimulating debates held during Summer 2007 and Spring 2010. The author asks for the indulgence of the readers as many points still need to be clarified or corrected ; remarks and comments on this draft will be most welcome.

References

[1] Nino Boccara and Henryk Fuks, *Number-conserving cellular automaton rules*, Fundamenta Informaticae **52** (2002), no. 1-3, 1–13.

[2] Nazim Fatès, *Robustesse de la dynamique des systèmes discrets : le cas de l'asynchronisme dans les automates cellulaires*, Ph.D. thesis, École normale supérieure de Lyon, 2004.

[3] Nazim Fatès, Michel Morvan, Nicolas Schabanel, and Eric Thierry, *Fully asynchronous behavior of double-quiescent elementary cellular automata*, Theoretical Computer Science **362** (2006), 1–16.

[4] Henryk Fukś, *Nondeterministic density classification with diffusive probabilistic cellular automata*, Physical Review E **66** (2002), no. 6, 066106.

[5] P. Gacs, G. L. Kurdiumov, and L. A. Levin, *One-dimensional homogeneous media dissolving finite islands*, Problemy Peredachi Informatsii **14** (1987), 92–98.

[6] Mark Land and Richard K. Belew, *No perfect two-state cellular automata for density classification exists*, Physical Review Letters **74** (1995), no. 25, 5148–5150.

[7] Gina M. B. Oliveira, Luiz G. A. Martins, Laura B. de Carvalho, and Enrique Fynn, *Some investigations about synchronization and density classification tasks in one-dimensional and two-dimensional cellular automata rule spaces*, Electronic Notes in Theoretical Computer Science **252** (2009), 121–142.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 231–244

CAvium - Strengthening Trivium Stream Cipher Using Cellular Automata

Sandip Karmakar, Debdeep Mukhopadhyay, Dipanwita Roy Chowdhury

Department of Computer Science and Engineering,
Indian Institute of Technology, Kharagpur,
India

Cellular Automata configurations are known to be able to generate good pseudorandom sequences. Linear Cellular Automata and LFSRs are equivalent in pseudorandom sequence generation, but those structures could be easily cryptanalysed due to their lack of nonlinearity. It is noted in this paper that, introduction of both nonlinear and linear rules in Cellular Automata structures can reach the desired setup state of a cipher much faster than the LFSR and NFSR based contemporary systems and provides comparatively secure design. The **eStream** stream cipher Trivium, in spite of, being secure in its full round operation, till date, has a large number of cryptanalysis on reduced versions of it. Trivium also has a long key setup process. In the present paper, we present a modification of the Trivium stream cipher using Cellular Automata which is shown to be faster in operation and is much secure than the original cipher. The proposed modification also has a shorter key setup process than Trivium.

Keywords: Cellular Automata, Trivium, Stream Cipher, Strengthening Trivium, Cryptography, Cryptography using Cellular Automata

1 Introduction

Cellular Automata are self-evolving systems of cells each of which updates itself per cycle following a rule embedded into it. Cellular Automaton (CA) is known for its ability to generate pseudorandom sequences needed for various applications like VLSI testing and coding theory, Wolfram (1986). Several researchers have attempted to apply the pseudorandomness of CA to cryptography. The cryptanalysis of linear CA based cryptographic techniques, Paterson et al. (1997) show that nonlinearity is needed for cryptographic applications. A 3-neighbourhood nonlinear CA each of whose cells updates itself by the nonlinear rule 30 has long been considered a very good pseudorandom sequence generator. It passed various statistical tests for pseudorandomness with good results, Wolfram (1985), until Willi Meier and Othmar Staffelbach proposed an attack on pseudorandom sequences generated by rule 30 CA, Meier and Staffelbach. (1991), which would break any such system of 300 cells in complexity of about 2^{19} operation. Other attacks on rule 30 CA were also reported, Koc and Apohan. (1997). These findings show that for cryptography, the data stream generated by CA needs to satisfy additional properties. In this paper, we use CA in connection with feedback nonlinearities as structured in the Trivium stream cipher to illustrate the use of CA in cipher design.

Trivium, introduced in the **eStream** project is among the top 5 stream ciphers at the end of the project. The cipher is very simple in design yet quite secure in operation. Although Trivium is still secure in its full round of operation, it is not secure in its reduced round versions. A very small growth rate in algebraic degree has lent Trivium to various trivial attacks like algebraic attacks, statistical attacks and higher order differential attacks in reduced rounds. The weakness rendered in Trivium due to the small growth rate of essential cryptographic properties is compensated by its long key setup process, which completes in 1152 cycles of operation. After the setup process, the cipher represents a secure structure against the cryptanalysis methods till encountered. However, as mentioned, a number of cryptanalytic results on reduced round Trivium are known. Linearization, correlation attacks, algebraic attacks were reported immediately after the introduction of the cipher. The proposal of AIDA has proved to be the strongest form of attack against Trivium. Till date AIDA can recover up to 793 reduced rounds of Trivium and Cube testers can distinguish up to 885 rounds of Trivium from a random sequence. Scan based side channel attack presented a cryptanalysis on full operational Trivium hardware. Though the cipher is still considered safe from a full operational attack, certain modifications on the cipher both to strengthen against reduced round attacks and also to speedup the setup process of the cipher can be proposed. The aim of the modifications would be to accelerate growth rate of essential cryptographic properties of a cipher system.

In this paper, we propose a CA based modification of the Trivium stream cipher which strengthens it against almost all the attacks encountered against reduced round Trivium so far. The modified cipher also provides a faster key setup process. The modified cipher will be called CAvium. We show quantitatively that CAvium is resistant against linear and higher order differential cryptanalysis and also resists correlation and algebraic attack. It is also demonstrated to be safe against scan-based side channel attacks. A hardware and software performance comparison of CAvium with Trivium is also given.

This paper is organized as follows. Following the introduction, section 2 discusses preliminaries. In section 3, we present a brief description of Trivium and list known attacks on it. CAvium is proposed in section 4. Section 5 analyzes the security strength of the proposed cipher. Performance of CAvium against existing attacks is presented in section 6. An overall comparison of CAvium and Trivium is presented in section 7. The paper is concluded in section 8.

2 Preliminaries

In this section, we present the basic terminology used in this paper.

A variable or its negation (x or $\bar{x}$) is called a *literal*. Any number of 'and'-ed literals is called a *conjunction*. For example, $x.y.\neg z$ is a *conjunction*.

Definition 1 *Algebraic Normal Form: Any* Boolean function *can be expressed as* XOR *of* conjunctions *and a* Boolean constant*,* True *or* False*. This form of the* Boolean function *is called its* Algebraic Normal Form (ANF)*.*

Definition 2 *Balanced Boolean Function: If the Hamming weight of a Boolean function of n variables is 2^{n-1}, it is called a balanced Boolean function.*

Thus, $f(x_1, x_2) = x_1 \oplus x_2$ is balanced, while $f(x_1, x_2) = x_1.x_2$ is not balanced.

Definition 3 *Nonlinearity: Let, f be a Boolean function of variables, $x_1, x_2, \ldots x_n$ and A be the set of all affine functions in $x_1, x_2, \ldots x_n$. The minimum of the Hamming distances between f and the Boolean functions in A is the nonlinearity of f.*

Hence, nonlinearity of $f(x_1, x_2) = x_1.x_2$ is 1.

Definition 4 *Walsh Transform: Let $\bar{X}$ = $(X_n, \ldots, X_1)$ and $\bar{\omega}$ = $(\omega_1, \ldots \omega_n)$ both belong to $\{0,1\}^n$ and $\bar{X}.\bar{\omega} = X_n.\omega_1 \oplus \ldots X_1.\omega_n$. Let $f(\bar{X})$ be a Boolean function on n variables. Then the Walsh transform of $f(\bar{X})$ is a real valued function over $\{0,1\}^n$ that can be defined as $W_f(\bar{\omega}) = \Sigma_{\bar{X} \in \{0,1\}^n}(-1)^{f(\bar{X}) \oplus \bar{X}.\bar{\omega}}$. The Walsh transform is sometimes called the spectral distribution or simply the spectrum of a Boolean function.*

Definition 5 *Resiliency: A function $f(X_n \ldots X_1)$ is m-th order correlation immune (CI)* iff *its Walsh transform W_f satisfies $W_f(\bar{\omega}) = 0$; for $1 \leq wt(\bar{\omega}) \leq m$. Further, if f is balanced then $W_f(0) = 0$. Balanced m-th order correlation immune functions are called m-resilient functions. Thus, a function $f(X_n, \ldots, X_1)$ is m-resilient iff its Walsh transform W_f satisfies $W_f(\bar{\omega}) = 0$; for $0 \leq wt(\bar{\omega}) \leq m$.*

For example, resiliency of $f(x_1, x_2) = x_1 \oplus x_2$ is 1, but resiliency of $f(x_1, x_2) = x_1.x_2$ is 0.

d-Monomial test is a statistical test for pseudorandomness introduced independently in Filiol. (2002) and Saarinen. (2006). It investigates the Boolean function representation of each output bit in terms of input bits. If a Boolean function of n Boolean variables is a good pseudorandom sequence generator, then it will have $\frac{1}{2}\binom{n}{d}$ d-degree monomials. The distribution is *binomial*. A χ^2 test with one degree of freedom is applied to count to measure how *unbiased* the count is. A deviation will indicate non-randomness. For example, consider the function $f(x_1, x_2, x_3) = x_1 \oplus x_2$, it has 2, 1-degree monomials and 0, 2 degree monomial. The ideal number of 1, 2 and 3 degree monomials would be $\frac{1}{2}\binom{3}{1}$ = 1.5, $\frac{1}{2}\binom{3}{2}$ = 1.5 and $\frac{1}{2}\binom{3}{3}$ = 0.5. It turns out that it has 2, 1-degree monomials more and 1 2-degree monomial less, hence it is expected to be non-pseudorandom. On the other hand, $f(x_1, x_2, x_3) = x_1 \oplus x_2.x_3$ is expected to be a good pseudorandom generator.

Definition 6 *Cellular Automata: A cellular automaton is a finite array of cells. Each cell is a finite state machine $C = (Q, f)$ where Q is a finite set of states and f a mapping $f : Q^n \rightarrow Q$. The mapping f, called local transition function. n is the number of cells the local transition function depends on. On each iteration of the CA each cell of the CA updates itself with respective f.*

The number of neighbouring cells, f depends on, may be same or different on different directions of the automaton. f may be same or different for cells across the automaton. The array of cells may be multi-dimensional. Hence, a huge number of CA configurations are possible. In this paper, we model rules as Boolean functions, so that, $Q = \{0, 1\}$. Each cell of the system is initialized with a Boolean value. Collectively, over the automaton it is referred to as the *seed*.

Definition 7 *Dimension: Dimension of the cell array is called the dimension of the CA.*

In this paper, we have considered 1-dimensional CA only.

Definition 8 *Neighbourhood: Adjacent cells of a cell are called the neighbourhood of CA.*

A 1-dimensional CA, each of whose rule depends on left and right neighbour and the cell itself is called a 3-neighbourhood CA. Similarly, if each cell depends on 2 left and 2 right neighbours and itself only, it is called 5-neighbourhood CA. A CA whose cells depend on 1 left and 2 right neighbouring cells is called a 4-neighbourhood right skew CA. A left skewed 4-neighbourhood CA can be defined similarly.

Definition 9 *Rule:The local transition function for a 3-neighbourhood CA cell can be expressed as follows:*

$$q_i(t+1) = f[q_i(t), q_{i+1}(t), q_{i-1}(t)]$$

where, f denotes the local transition function realized with a combinational logic, and is known as a rule of CA, Nandi et al. (1997). The decimal value of the truth table of the local transition function is defined as the rule number *of the cellular automaton.*

For example, for 1-dimensional 3-neighbourhood CA,
Rule 30: $f = q_{i-1}(t) \oplus (q_{i+1}(t) + q_i(t))$, where + is the Boolean 'or' operator and $\oplus$ is the Boolean 'XOR' operator.
Rule 60: $f = q_{i-1}(t) \oplus q_i(t)$.
Rule 90: $f = q_{i-1}(t) \oplus q_{i+1}(t)$.

Definition 10 *Uniform Cellular Automaton: A CA whose local transition function is same for all the cells is called uniform cellular automaton.*

Definition 11 *Hybrid Cellular Automaton: A CA whose local transition function is* not *same for all the cells is a hybrid cellular automaton.*

Definition 12 *Linear Cellular Automaton: A CA whose local transition function ANF does not involve the '.' (Boolean and) operator in any of the cell is called the linear cellular automaton. For example, rule, $f = q_{i-1}(t) \oplus q_{i+1}(t)$ employed in each cell is a linear cellular automaton, where $q_{i-1}(t)$ and $q_{i+1}(t)$ denotes left and right neighbours of i-th cell at t-th instance of time.*

Definition 13 *Nonlinear Cellular Automaton: A CA whose local transition function is non-linear, i.e., involves at least one . (Boolean and) operator, for at least one of the cells is a nonlinear cellular automaton. For example, rule, $f = q_{i-1}(t).q_{i+1}(t)$ employed in each cell is a nonlinear cellular automaton, where, $q_{i-1}(t)$ and $q_{i+1}(t)$ denotes left and right neighbours of the i^{th} cell at t^{th} instance of time.*

Any CA can be utilized to generate pseudorandom sequences of different degree of security by first selecting a *seed* and then updating each cell according to the transition functions. State values from the middle cell of the cell array may be taken output to represent generation of pseudorandom sequences.

3 Trivium

In this section, we present the algorithm of Trivium operation. We also brief the known attacks against Trivium.

3.1 Description of Trivium

Trivium was introduced in the **eStream** cipher project in, Canniere and Preneel (b) by Christophe De Cannière and Bart Preneel. It is a *synchronous* stream cipher. Developers focused on a block cipher based design principles while constructing Trivium, Canniere and Preneel (a). The cipher works on 80-bit *secret* key and 80-bit *public* initial vector (IV). Hence, the expected security strength of the cipher is 2^{80}. It takes 1152 rounds to *initialize* itself. *Key stream bits* are output only after this *initialization* phase is over. Once initialized, it can produce up to $N = 2^{64}$ pseudorandom key stream bits. We give a short description of operation of Trivium below. A detailed description can be found in the **eStream** website.

Trivium consists of 288 internal 1-bit state registers, $(s_1, s_2, \ldots, s_{288})$. Operationally, the registers are organized as three right shift registers of lengths 93, 84 and 111 bits, respectively. These three registers are, *key bit registers*, *IV bit registers* and *constant bit registers*. As already mentioned, the cipher operates in

two phases, *initialization* and *key stream generation*. Both the phases however execute the same algorithm. The algorithm is presented next.

$$\begin{aligned}
\textbf{for } i &= 1 \textbf{ to } N \textbf{ do} \\
t_1 &= s_{66} + s_{93} \\
t_2 &= s_{162} + s_{177} \\
t_3 &= s_{243} + s_{288} \\
z_i &= t_1 + t_2 + t_3 \\
t_1 &= t_1 + s_{91}.s_{92} + s_{171} \\
t_2 &= t_2 + s_{175}.s_{176} + s_{264} \\
t_3 &= t_3 + s_{286}.s_{287} + s_{69} \\
(s_1, s_2, \ldots, s_{93}) &\leftarrow (t_3, s_1, s_2, \ldots, s_{92}) \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (t_1, s_{94}, s_{95}, \ldots, s_{176}) \\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (t_2, s_{178}, s_{179}, \ldots, s_{287}) \\
\textbf{end for}
\end{aligned}$$

The registers t_1, t_2 and t_3 are temporary registers and z_i is the i^{th} output *key stream bit*.
During *initialization*, the above algorithm is executed for $4 \times 288 = 1152$ cycles. But, this phase *does not* output any key stream bit, z_i.

80-bit key, $\{k_1, k_2, \ldots, k_{80}\}$ and 80-bit IV, $\{iv_1, iv_2, \ldots, iv_{80}\}$ are loaded in the internal state registers as follows:

$$\begin{aligned}
(s_1, s_2, \ldots, s_{93}) &\leftarrow (k_1, k_2, \ldots, k_{80}, 0, \ldots, 0) \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (iv_1, iv_2, \ldots, iv_{80}, 0, \ldots, 0) \\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (0, 0, \ldots, 0, 1, 1, 1)
\end{aligned}$$

The *key stream generation* phase updates internal state registers according to the algorithm. The *key stream bit*, z_i is output at each cycle in this phase.

3.2 Weaknesses of Trivium

A number of cryptanalytic results on Trivium stream cipher is known. Though almost all of them are on reduced round versions of Trivium, those nevertheless, demonstrate weaknesses of the cipher. Till date 793 rounds of Trivium could be cryptanlyzed by recovering key with less than brute-force complexity and 885 rounds of Trvium can be distinguished from a random sequence. Most notably, higher order differential attack, AIDA happened to be vastly successful against the cipher. Table 1 lists the significant attacks reported on Trivium.

We would briefly discuss the main reasons of success of the mentioned attacks.

Tab. 1: Trivium:Known Attacks

Attack	*Rounds*	*Distinguisher/Key Recovery*
Linearization, Turan and Kara	288	Key Recovery
Correlation Attack, Maximov and Biryukov	Bivium	Distinguisher
Algebraic Attack, McDonald et al.	Bivium without Setup phase	Key Recovery
Scan-Attack, Agarwal et al.	Full Trivium	Key Recovery
AIDA (or Cube Attack), Vielhaber (2007)	793	Key Recovery
Cube Tester, Aumasson et al. (2009)	885	Distinguisher

- Linearization Attack: Linearization of a cipher is possible only when a cipher does not grow faster in nonlinearity. 288 rounds of Trivium could be linearized implies lack of such nonlinearity growth. A quantitative measure of nonlinerity of Trivium is given in table 3. The lack of enough growth is due to low nonlinearity addition with iterations.

- Algebraic Attack: Full Bivium without setup phase can be analyzed directly using SAT-solver. This is again mainly due to addition of low nonlinearities with iterations.

A CA based modification is hence proposed here in view of preventing against the above reported attacks even on reduced round versions of Trivium.

4 CAvium Proposal

CAvium is a CA based modification of the Trivium stream cipher. Basically, CAvium replaces only the Shift Register of Trivium with a hybrid $< 30, 60, 90, 120, 150, 180, 210, 240 >$ CA. Here, $< 30, 60, 90, 120, 150, 180, 210, 240 >$ hybrid CA means rule 30, rule 60, ..., rule 240 CA cells placed alternatively. The basic principle of the design is utilization of parallelization of CA combined with essential crypto-properties of it.

Like Trivium, CAvium also consists of 288 internal 1-bit state registers, $(s_1, s_2, \ldots, s_{288})$. Register s_1 operates on CA rule 30, register s_2 operates on CA rule 60 etc. Register s_8 has rule 240 embedded and again register s_9 operates on rule 30 and so on. So, there are 36 repetitions of each of the CA rules over the 288 bit register. Operationally, the registers are organized as three hybrid $< 30, 60, 90, 120, 150, 180, 210, 240 >$ CA of lengths 93, 84 and 111 bits, respectively. These three CAs are, *key bit CA*, *IV bit CA* and *constant bit CA* respectively. In this construction, we have however dropped the discontinuities among the registers, so for example, s_{93} and s_{94} are adjacent to each other. Thus for example, neighbouring cells of s_{94} are s_{93} and s_{95}. Similar is the case with s_{93}, s_{177} and s_{178}. Like Trivium, this cipher also operates in two phases, *initialization* and *key stream generation*. Both the phases as before execute the same algorithm. The algorithm is depicted next.

$$\begin{aligned} \textbf{for } i &= 1 \textbf{ to } N \textbf{ do} \\ t_1 &= s_{66} + s_{93} \\ t_2 &= s_{162} + s_{177} \\ t_3 &= s_{243} + s_{288} \end{aligned}$$

$$
\begin{aligned}
z_i &= t_1 + t_2 + t_3 \\
t_1 &= t_1 + s_{91}.s_{92} + s_{171} \\
t_2 &= t_2 + s_{175}.s_{176} + s_{264} \\
t_3 &= t_3 + s_{286}.s_{287} + s_{69} \\
(s_1, s_2, \ldots, s_{93}) &\leftarrow (t_3, CA(s_1), CA(s_2), \ldots, CA(s_{92})) \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (t_1, CA(s_{94}), CA(s_{95}), \ldots, CA(s_{176})) \\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (t_2, CA(s_{178}), CA(s_{179}), \ldots, CA(s_{287}))
\end{aligned}
$$

end for

Here, $CA(s)$ refers to the Boolean value obtained in the cell s upon operation in $< 30, 60, 90, 120, 150, 180, 210, 240 >$ hybrid CA rule.

The registers t_1, t_2 and t_3 are temporary registers and z_i is the i^{th} output *key stream bit*.

During *initialization* the above algorithm is executed for $4 \times 36 = 144$ cycles. But, this phase *does not* output any key stream bit, z_i. 80-bit key, $\{k_1, k_2, \ldots, k_{80}\}$ and 80-bit IV, $\{iv_1, iv_2, \ldots, iv_{80}\}$ are loaded in the internal state registers as follows:

$$
\begin{aligned}
(s_1, s_2, \ldots, s_{93}) &\leftarrow (k_1, k_2, \ldots, k_{80}, 0, \ldots, 0) \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (iv_1, iv_2, \ldots, iv_{80}, 0, \ldots, 0) \\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (0, 0, \ldots, 0, 1, 1, 1)
\end{aligned}
$$

The *key stream generation* phase updates internal state registers according to the algorithm. The *key stream bit*, z_i is output for each cycle at this phase.

5 Cryptographic Properties of CAvium

In this section, we describe the cryptographic advantages we obtain with our modification of the stream cipher Trivium. The following properties are known to be important for security of ciphers:

1. Balancedness
2. Nonlinearity
3. Correlation Immunity
4. Algebraic Degree
5. d-monomial test

Nonlinearity and correlation immunity are the most important requirements among the first four. Good nonlinearity characteristics indicate that the cipher is expected to be safe against linear cryptanalysis and

Tab. 2: CAvium:Cryptographic Characteristics of the Output Bit

Iteration	*Balancedness*	*Nonlinearity*	*Algebraic Degree*	*Resiliency*
1	Balanced	0	1	1
2	Balanced	0	1	3
3	Balanced	384	2	5
4	Balanced	1792	3	6

Tab. 3: Trivium:Cryptographic Characteristics of the Output Bit

Iteration	*Balancedness*	*Nonlinearity*	*Algebraic Degree*	*Resiliency*
1	Balanced	0	1	1
70	Balanced	16	2	3
71	Balanced	32	2	3
83	Balanced	384	3	4
98	Balanced	1792	3	5

also from algebraic attacks. However, good nonlinearity characteristics does not imply correlation immunity, ie, good nonlinear ciphers can display correlations among key, plaintexts and ciphertexts. Hence, a fair mix of nonlinearity and correlation immunity is required. Algebraic degree characteristics are also important for resistance against algebraic attacks. Below we show the characteristics of the above properties for the CAvium stream cipher. A comparison with corresponding characteristics of Trivium is also given.

5.1 Balancedness

Table 2 illustrates the balancedness property of the CAvium output bit with iterations. All the output bit expressions are balanced in the initial 4 iterations. Table 3 illustrates the balancedness property of Trivium output bit expression with iterations. With respect to balancedness property both the ciphers generate balanced functions as output in the considered rounds.

5.2 Nonlinearity

Table 2 shows the nonlinearity of CAvium with pass of iteration and table 3 shows comparable nonlinearities of Trivium stream cipher. In only 4 cycles of operation nonlinearity of CAvium reaches a nonlinearity of 1792 which is reached by Trivium output bit at iteration 98. This high growth rate of nonlinearity guarantees protection against linear cryptanalysis.

5.3 Algebraic Degree

Table 2 shows the growth of algebraic degree of the output bit of CAvium with iterations. It can be observed that in CAvium the algebraic degree increases almost linearly. Table 3 lists the minimum number of iterations required in case of Trivium at which algebraic degrees 2 and 3 are reached. Clearly, CAvium has a much faster growth rate of algebraic degree compared to Trivium. Ciphers having large algebraic degrees are resistant against linearization and algebraic attacks. So, CAvium is expected to be stronger than Trivium with respect to these attacks both in reduced round version and the full key-IV setup version.

Tab. 4: CAvium:d-monomial Test Result Output Bit

Iteration	*Deg.-1*	*Deg.-2*	*Deg.-3*	*Deg.-4*	*Deg.-5*	*Deg.-6*
1	2	0	0	0	0	0
2	4	0	0	0	0	0
3	6	2	0	0	0	0
4	6	8	2	0	0	0
5	7	14	12	6	0	0
6	8	14	36	32	8	2

Tab. 5: Trivium:d-monomial Test Result Output Bit

Iteration	*Deg.-1*	*Deg.-2*	*Deg.-3*	*Deg.-4*
1	2	0	0	0
70	4	1	0	0
161	16	11	2	0
239	29	65	38	1

5.4 Resiliency

Table 2 tabulates the resiliency of CAvium output bit with iterations and resiliency of Trivium output bit is given in table 3. Those tables reveal that higher resiliency is achieved by CAvium at a much lower number of iterations, for example, Trivium output bit achieves resiliency 5 at iteration 85 while CAvium reaches it at iteration 3. Due to the faster growth of resiliency of output bit of CAvium, it is expected to show resistance against correlation attacks.

5.5 d-monomial Test

d-monomial test proposed independently in, Filiol. (2002) and Saarinen. (2006) is a statistical test for measuring randomness of ciphers. The test compares the closeness of the number of d^{th} degree n variable terms with the expected ideal number of d^{th} degree n variable terms of a truly random Boolean function. An ideal random Boolean function will have $\frac{1}{2} \times \binom{n}{d}$ d degree terms. We tabulate in table 4 the d-monomial test values for the first 6 iterations of the output bit of CAvium.

The growth in number of terms in the resultant Boolean expression and the number of different degree terms in the output equation are both high. This kind of distribution is expected to resist higher order differential attacks and distinguishers. In table 5 we tabulate the d-monomial test result of the Trivium at the iterations 1, 70, 161 and 239, i.e. at iterations where algebraic degrees are incremented. Note that, Trivium has a better diffusion of various degree terms compared to CAvium.

Considering table 4 once again, note that, at iteration 6 only the number of nonlinear terms in the expression of the output bit is more than 90, which is more than double the number of nonlinear terms at iteration 5, it can be expected that any attempt to linearize the expression for algebraic attack will have to deal with exponential number of nonlinear terms with pass of iterations. Hence, algebraic attacks are not expected to yield good result against CAvium.

If we compare over all the properties we have experimented against CAvium stream cipher, we can see that CAvium reaches the desired property values at very small number of iterations. Hence, the reduction in number of cycles required to initialize the cipher is not expected to leave any weakness in CAvium.

6 Performance against Existing Attacks

In this section we reason that the number of cryptanalytic results demonstrated against reduced versions of Trivium may not be successful against CAvium.

1. *Linear Cryptanalysis*: Linearization attack, Turan and Kara and linear circuit approximation, Khazaei and Hassanzadeh were demonstrated against Trivium in 288 round reduced version. Linear cryptanalysis on CAvium will not be successful because :
   - The high growth rate of algebraic degrees of CAvium (refer table 2).
   - Table 2 shows the growth of nonlinearity of the output bit of CAvium with iterations. It can be noted that the growth rate of nonlinearity is much steeper than Trivium.
   - Table 4 indicates that the number of linear terms also increase with iterations. As linear terms add exponentially to the nonlinearity growth of a Boolean expression, linearization is not expected to work.

   Hence, linear cryptanalysis will not be successful on CAvium.

2. *Algebraic Attacks*: Algebraic attack, McDonald et al. using SAT-solver was reported on reduced version of Trivium called Bivium-A. Algebraic cryptanalysis is dependent on the algebraic degree of a cipher. The increase of number of nonlinear terms of a cipher also increase the attack complexity. So, the high algebraic degree growth rate and exponential increase in number of nonlinear terms will prevent algebraic attacks on CAvium.

3. *Scan-based Side Channel Attack*: A scan-based side channel attack was reported on full round of Trivium on hardware implementation, Agarwal et al.. Scan-chain based attack on Trivium worked because of the invertibility of the states of the cipher. The same will not be possible for CAvium because of the presence of non-invertible CA rule 30. Though rule 30 is partially reversible, presence of linear rules in the CA configuration reduces the probability of the reversion exponentially with iterations. Hence, scan-based side channel attack will not be successful on CAvium.

4. *Cube Attack/AIDA attack*: Till date the most successful attacks on reduced round versions of Trivium were cube attacks (or AIDA attacks). This attack exploits the fact that the distribution of the d-degree terms is deviant from ideal in d-monomial test. A large *algebraic degree* of a cipher will prevent the attack from practically being implemented. Also, the result of *d-monomial test* of CAvium is much better than Trivium. The density of d-degree terms is though far from ideal in case of CAvium also; the closeness with ideal values is better than Trivium. The increase in different degree nonlinear terms with iterations are pretty fast and closer to ideal compared to Trivium. For example, in CAvium between round 5 and 6, degree 3 and degree 4 terms increase by 24 and 26 respectively which is close to ideal. Note that, Trivium has better growth in linear terms with iterations compared to CAvium (table 4). But, ideal values in higher degree terms are more important in resistance against AIDA than low degree terms. Hence, *cube attack* on CAvium is also expected *not* to be successful on any reasonable number of rounds on CAvium.

7 CAvium vs. Trivium

In this section, we enumerate the advantages and disadvantages of CAvium over Trivium.

1. Firstly, the startup process is made about 10 times faster over Trivium. Trivium stream cipher takes 1152 cycles of operation to get ready to produce key stream bits. Though after the initial setup, Trivium is able to generate 2^{64} key stream bits - for small and even moderately large encryptions, 1152 cycles only for key and IV setup is large. CAvium, on the other hand, takes 144 clock cycles to complete key and IV setup. Keystreams are generated from 145^{th} clock cycle only. After key and IV setup, CAvium can also generate 2^{64} key streams. Hence, the introduction of CA in Trivium design makes the cipher operation faster and suitable for even small length encryptions.

2. The small Cellular Automata based nonlinearity insertion in Trivium has led to a wide range of key recovery and distinguisher attacks on the cipher including linearization, correlation attacks and algebraic attacks. As we have shown in the table above, CAvium has a much steeper nonlinearity growth rate than Trivium. Also, CAvium has a large number of linear terms in its Boolean expression due to the presence of linear CA rules in its configuration. The presence of linear terms mandates inclusion of those rules in the linear approximation and the other nonlinear terms decreases linearization bias. Altogether, linearization bias of CAvium is much lower than Trivium even for reduced versions of it. Again, due to the construction of CAvium the increase in number of nonlinear terms of the output bit, $z = t1 + t2 + t3$ is exponential.

3. The proposed construction performs better in d-monomial test than Trivium. Hence, higher order differential (e.g., AIDA) key recovery attacks and distinguishers would be difficult for CAvium. In comparison, we have known a large number of higher order differential attacks on Trivium.

4. Due to the nature of CA construction, we know that, after t cycles of operation, a CA cell depends on $2t + 1$ neighboring cells. So, after 55 clocks of operation, every cell is dependent on every other cell of the 288 bit state register. Therefore, the highest nonlinearity is propagated to the other cells in at most 55 cycles. Note that, 144 cycles is the time required to set up key and IV hence the highest nonlinearity is propagated to all the cells at least twice. It also means that algebraic degree grows exponentially at most after 55 cycle of operation.

5. Though we have not given any theoretical proof of cycle length of CAvium, our experiment suggests a minimum of about 2^{64} cycle length is possible. However, it is a open problem whether there is any cycle of length of $< 2^{64}$ in CAvium.

6. The algebraic degree growth rate of CAvium is also quite high. If a simple linearization technique is used in CAvium for algebraic attack, even after 15 clock cycles, there will be more than 100 new variables. Hence, possible algebraic attacks will also not be plausible in case of CAvium.

7. Correlation of the CA structure is reduced due to the introduction of linear rules in the CA register. The measure presented above shows that correlation decreases exponentially with iteration, which is also a triumph over the original Trivium design, in which due to the presence of linear shift register, correlation does not decrease exponentially with number of cycles of operation. Hence, though a uniform rule 30 CA, Meier and Staffelbach. (1991) may be susceptible to correlation attack, CAvium structure would be safe against any correlation attacks.

8. Knowing the full state of the Trivium stream cipher after any cycle of operation could reveal the key of the cipher. This is due to the reversible nature of the Trivium nonlinear equations. This could lead to scan chain based attacks as depicted in, Agarwal et al.. The knowledge is however of little relevance for CAvium, since the presence of nonlinear non-invertible rule 30 in the CA operation prevents any such inversion. Further though rule 30 is partially invertible, the presence of linear rules along with the nonlinear rules reduces the probability of inversion exponentially as the correlation decreases. Hence, a scan based attack with non-secure scan chains can not also break the system.

8 Conclusion

In the current paper, we have presented a modification of the eStream stream cipher winner Trivium, Canniere and Preneel (b). The proposed modified cipher has comparatively better nonlinearity, algebraic degree, resiliency characteristics than Trivium. Also, CAvium achieves the mentioned characteristics faster than Trivium, as fast as, 10 times than Trivium. The faster growth rate of cryptographically essential properties has also helped CAvium to reduce the longer setup process of Trivium, so that, CAvium takes only 144 clock cycles of operation to complete setup compared to 1152 clock cycles taken by normal Trivium operation. It is also reasoned that the proposed modification is expected to be resistant against attacks such as linearization, algebraic cryptanalysis, correlation attacks, scan-based side channel attacks and higher order differential attacks such as AIDA (or Cube attack) even in reasonable reduced round versions. Though not explicitly mentioned the d monomial characteristics of CAvium demonstrates that differential attacks also may not be successful against CAvium. Again extensive experiment is conducted to determine small cycle lengths of CAvium but we expect a cycle length of $< 2^{64}$ may not be possible, though no theoretical explanation is provided. The operation of CAvium could be a bit slower than Trivium. Hence, CAvium is a secure modification of Trivium with much faster setup.

The construction of CAvium is based on an underlying mixed CA consisting of linear and nonlinear rules. The fast growth of nonlinearity, resiliency, algebraic degree is attributed mainly to the presence of nonlinear rule (rule 30). In essence the only Trivium specific operation included in CAvium is the introduction of three nonlinear terms per cycle of operation. A comparison of d-monomial characteristics of Trivium and CAvium will show that the essential growth in nonlinearity and algebraic degree is due to the presence of nonlinear rule 30. So in fact a plain construction of a cipher consisting only of the 288 bit CA may be a good cipher candidate for study. It can be mentioned that a good d-monomial characteristic can be the one and only desirable property of a good cipher. Hence, construction of stream ciphers using CA as the *only* building block is an interesting area of study.

References

M. Agarwal, S. Karmakar, D. Saha, and D. Mukhopadhayay. Scan-based side channel attack on stream ciphers and its countermeasures. *INDOCRYPT, 2008*.

J.-P. Aumasson, I. Dinur, W. Meier, and A. Shamir. Cube testers and key recovery attacks on reduced-round md6 and trivium. In *FSE*, pages 1–22, 2009.

C. D. Canniere and B. Preneel. A stream cipher construction inspired by block cipher design principles. *eSTREAM, ECRYPT Stream Cipher Project*, 2006, a.

C. D. Canniere and B. Preneel. Trivium specifications. *eSTREAM, ECRYPT Stream Cipher Project*, 2006, b.

E. Filiol. A new statistical testing for symmetric ciphers and hash functions. *Proc. Information and Communications Security 2002, Volume 2513 of LNCS*, pages 342–353, 2002.

S. Khazaei and M. Hassanzadeh. Linear sequential circuit approximation of the trivium stream cipher. *eSTREAM, ECRYPT Stream Cipher Project*, 2005.

C. Koc and A. Apohan. Inversion of cellular automata iterations. *Computers and Digital Techniques, IEE Proceedings - Volume - 144, Issue: 5*, pages 279–284, 1997.

A. Maximov and A. Biryukov. Two trivial attacks on trivium. *eSTREAM, ECRYPT Stream Cipher Project*, 2007.

C. McDonald, C. Charnes, and J. Pieprzyk. Attacking bivium with minisat. *eSTREAM, ECRYPT Stream Cipher Project*, 2007.

W. Meier and O. Staffelbach. Analysis of pseudo random sequences generated by cellular automata. *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1991.

S. Nandi, S. Chattopadhyay, P. P. Chaudhuri, and D. R. Chowdhury. Ca and its applications: A brief survey. In *Additive Cellular Automata - Theory and Applications*, volume 1, 1997.

K. G. Paterson, S. R. Blackburn, and S. Murphy. Theory and applications of cellular automata in cryptography. In *IEEE Transactions on Computers, Volume 46, Issue 5*, 1997.

M. J. O. Saarinen. Chosen-iv statistical attacks on e-stream stream ciphers. *eSTREAM, ECRYPT Stream Cipher Project, Report 2006/013*, pages 5–19, 2006.

M. S. Turan and O. Kara. Linear approximations for 2-round trivium. *eSTREAM, ECRYPT Stream Cipher Project*, 2007.

M. Vielhaber. Breaking one.fivium by aida an algebraic iv differential attack. Cryptology ePrint Archive, Report 2007/413, 2007. "http://eprint.iacr.org/".

S. Wolfram. Cryptography with cellular automata. In *CRYPTO: Proceedings of Crypto*, 1985.

S. Wolfram. Random sequence generation by cellular automata. In *Advances in Applied Mathematics, Volume-7*, pages 123–169, 1986.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 245–254

Multilane Single GCA-w Agent-based Expressway Traffic Model

Anna T. Lawniczak[1 3] and Bruno N. Di Stefano[2 3]

[1] *Department of Mathematics & Statistics, University of Guelph,*
50 Stone Road East, MacNaughton Building, Guelph, ON Canada N1G 2W1
alawnicz@uoguelph.ca
[2] *Nuptek Systems Ltd*
Toronto, Ontario, Canada M5R 3M6
b.distefano@ieee.org, Bruno.DiStefano@nupteksystems.com
[3] *The Fields Institute for Reserach in Mathematical Sciences,*
222 College Street, Toronto, Ontario M5T 3J1, Canada

We presents a brief description of a stream of research on highway and expressway traffic modeling and simulation. A first model has been developed starting from ECA Rule 184 and applying concepts similar to the ones employed in the Nagel Schreckenberg model. The second model is based on the "Global Cellular Automata"(GCA) and the "Global Cellular Automata with Write access"(GCA-w) developed by Rolf Hoffmann and his collaborators. This allows for faster execution and for the elimination of some potential conflicts during execution. We present an example of digital experiment that can be used to help traffic engineers in deciding the topology of entry and exit ramps on an expressway.

Keywords: Cellular Automata, Global Cellular Automata, Global Cellular Automata with Write access, Highway Traffic Modeling, Expressway Traffic Modeling.

1 Introduction

Depending on jurisdiction where one resides, the same type of road may be called an "expressway"or a "freeway". We refer to the term "expressway"to indicate a divided highway for high-speed vehicular traffic with controlled access, via entry & exit ramps, and no intersections at grade. We are conducting research, by modeling and simulation, on the effects of flow and congestion on "travel time"through a realistic long expressway. "Travel time"is "the total time required for a vehicle to travel from one point to another over a specified route under prevailing conditions", [1]. The difference between our work and published research previously conducted by others is that we aim to model much longer expressways, e.g. at least 1000 km, and a much higher number of vehicles, e.g. realistic traffic conditions over several days, e.g. a week. Additionally, all models we have seen do not have features that are very important from an engineering point of view, such as the ability to track individual vehicles during their trip, from entry ramp to exit ramp. We need this ability to simulate wireless vehicle tracking for only a small subset of vehicles.

We plan on using this model for practical traffic engineering applications. This paper is structured as follows: section 2 gives some introductory information about microscopic, individually based highway traffic models, section 3 contains the description of a multilane 2-D CA (Cellular Automata) agent based model previously developed by us, section 4 describes a multilane single GCA-w (Global Cellular Automata with Write access) agent based model developed by us and explains its computational advantages, section 5 presents and example of digital experiment that can be used to help traffic engineers in deciding the topology of entry and exit ramps on an expressway. Future planned work is described in section 6.

2 Microscopic, Individually Based Highway Traffic Models

Traditional macroscopic models of the 50s, such as the LWR model (Lighthill, Whitham, Richards), [2] and [3], are characterized by a large number of parameters without an immediately intuitive equivalent when conducting empirical investigations. Microscopic models based on cellular automata (CA) such as the one of Cremer and Ludwig, [4], and the Nagel Schreckenberg model, [5], have solved this problem. Their model can be seen as an extension of ECA (Elementary CA) Rule 184. This rule accurately describes the motion of a vehicle at constant speed of one cell per time step and null acceleration. This is unrealistic, but is a good starting point to apply extensions to Rule 184 as it may be needed. It is important to notice that Rule 184 is deterministic and cannot simulate real traffic with accidents. The Nagel Schreckenberg model solves the problem adding stochastic behaviour, a larger size neighbourhood that can be used to implement variable speed and non null acceleration. The Nagel Schreckenberg model is the origin of a very significant stream of research of derived research, see for instance [6], [7], [8], [9], and [10]. This stream is still uninterrupted and very rich of results.
However, most of this research has been conducted from the perspective of physics and statistical physics, to investigate dynamical aspects of highway traffic, see for instance [11] and [12]. Most such models lack features that may be required in some engineering applications, such as the ability to track individual vehicles during their trip, from entry ramp to exit ramp, knowing exactly on which cell a vehicle is at each time step. However, the Nagel Schreckenberg model is almost always the starting point of the design of new models.
Nagel and Schreckenberg proposed a stochastic model based on a neighbourhood of 5 cells and six discrete velocities. The model consists of four steps that have to be applied simultaneously to all cars:

- Acceleration
- Safety Distance Adjustment ("*slowing down due to other cars*")
- Randomization
- Change of Position

During the "Acceleration"phase, at each time step, if the velocity of the vehicle at the end of the previous time step is $v < v_{max}$, the velocity is incremented by one unit: $v- > v + 1$. If the velocity of the vehicle at the end of the previous time step is $v = v_{max}$, the velocity is left unchanged (null acceleration).
During the "Safety Distance Adjustment", if a vehicle has d empty cells in front of it and its velocity v, after the Acceleration phase, would cause the vehicle to cover a distance larger than d, then the vehicle decelerates, that is, it reduces its velocity to d: $v- > \min\{d, v\}$. If the d cells in front of the vehicle are empty, no deceleration is required.

As stated by Nagel and Schreckenberg, the randomization "*is essential in simulating realistic traffic flow since otherwise the dynamics is completely deterministic. It takes into account natural velocity fluctuations due to human behaviour or due to varying external conditions.*"[5]. Randomization is an extension to the traditional deterministic paradigm of ECA and can be found in all realistic highway traffic models based on CA. The "Change of Position"assumes that the new velocity v_n for each car n causes advancing by v_n cells: $x_n -> x_n + v_n$.
The implementation of this model requires to modify the CA paradigm and to make the evolution of the CA not only dependent on the state of the neighbourhood but also on the current velocity of each vehicle. This implies that each cell is characterized not only by presence or absence of a vehicle but also by a pointer to a data structure containing the current velocity of the vehicle. Here we do not use the word "pointer"in the sense of the C/C++ programming language, but in the sense of "link, connection". Almost all models that we have examined implement variable velocity as in the Nagel and Schreckenberg model, the only substantial difference being the number of cells that the vehicle may need to advance to achieve its maximum speed.
Nagel and Schreckenberg write that "*Through the steps one to four very general properties of single lane traffic are modeled on the basis of integer valued probabilistic cellular automaton rules.*"
We have perused the literature looking for ways in which others have handled multilane highway traffic. We have found 2-D CA implementations and Multi-CA implementations (i.e., one per lane). In the case of 2-D implementations the highway is represented by a CA consisting of a number of rows equal to the number of lanes being modeled and by a large number of cells representing the entire length of the highway. Lane changing is accomplished by simply moving to the adjacent cell on a different row. We developed a model of this type in the early stages of our research, [13]. We describe it in Section 3. Multi-CA implementations threat every CA as a separate road. The transition rules apply equally to every CA. Lane changing simply implies moving to the cell having the same cell number in the adjacent CA.

3 Our Multilane 2-D CA Agent Based Model

As a first step, we have developed a two dimensional CA expressway traffic model "capable of realistically simulating: a multi-lane expressway with multiple entry and exit ramps situated at various locations, vehicle following, speed increment up to maximum speed selectable on a per vehicle basis, lane change to pass or to go back to the right-most lane as it may be required by road rules in some jurisdictions, slowing down or stopping to avoid obstacles.", [13]. We represented the expressway by means of a CA consisting of a number of rows equal to the number of lanes being modeled and by a large number of cells representing the entire length of the expressway. Each cell was assumed to be 7.5 m long, as in most of the literature of microscopic, individually based highway traffic models, e.g. [5] and [14]. This has been chosen because it corresponds to the space occupied by the typical car plus the distance to the preceding car in a situation of dense traffic jam. The traffic jam density is given by 1000/7.5 m approximately equal to 133 vehicles per km,. We accomplished lane changing by simply moving to the adjacent cell on a different row, i.e. that is a cell with the same column number and a different row number, incrementing the row number when moving to the left and decrementing the row number when moving to the right. This model is characterized by variable size neighbourhood to implement safe driving distances. For every vehicle these safe driving distances are a function of the vehicle velocity and of where the vehicle is at a given moment during the simulation.
Traffic is modeled applying the same algorithm at each time step, when each cell of each lane is examined

in sequence and, if occupied by a vehicle, the vehicle navigation algorithm is applied. Each vehicle is an agent capable of deciding which action to take within a certain number of predefined actions. Implementation details can be found in [13].
Executing the software for a given configuration of the model means executing three nested loops. The external loop is the time loop, the next loop is the row loop, and the innermost loop is the column loop. At the end of each step of each loop the end of loop condition is tested by the software even if this is not explicitly evident to the user of the software.

3.1 Our Multilane Single GCA-w Agent Based Model

Thus, modeling traffic is equivalent to executing two large loops, an external time loop and an internal space loop. In reality, as we will see when describing the implementation the space loop is replaced by a number of loops where various operations are performed in sequence. Thus, after an initialization of all data structures used in the model, all execution time of the model is spent in these two loops. At each time step, vehicles are generated at each entry ramp according to a predefined vehicle generation probability that can be specified individually for each entry ramp. In the implementation, the software reads an input configuration file containing a description of the entire expressway. One of the predefined commands describes each entry ramps and the characteristics of the vehicles entering at that ramp. This command can be repeated multiple times for each entry ramp, indicating the different traffic characteristics at various times of the day (e.g., rush hour, day time, night time, work day, weekend, etc). For each instance of this command it is possible to specify:

- "Entry lane number"(always lane zero, that is the rightmost lane, except when entry cell is cell number zero, that is the entry to the expressway);
- "Entry cell number"(the location of the entry ramp from the beginning of the highway);
- "Start Time"and "End Time"measured in time steps from the beginning of the simulation when the specified creation probability applies;
- "Vehicle creation probability"during the specified time interval;
- Probability that the vehicle will be instantiated with the last cell of the expressway as its destination;
- "Maximum speed"that the vehicle will be able to reach while travelling on the expressway;
- Probability that the vehicle will be instantiated with a maximum speed equal to the one specified in the command.

The final destination probability and the maximum speed probability define not only the obvious probabilities implied by their names, but also the behaviour of the complementary probabilities. In other words if Pd is the probability that the vehicle is instantiated with last cell as its destination, $(1 - P_d)$ is the probability that the vehicle will go elsewhere, to other exit ramps. The specific exit ramp is assigned randomly. Similarly, if P_{vmax} is the probability that the vehicle will be instantiated with the specified maximum speed, $(1 - P_{vmax})$ is the probability that the vehicle will be instantiated with a different maximum allowable speed. The specific different speed will be assigned randomly.
After all vehicles have been generated for each entry ramp, they are queued and placed on the ramp data structure (a first-in-first-out queue).

At this point, each vehicle on the highway, represented by a different instance of an agent, executes its navigation algorithm, that is the algorithm allowing changing lane, if required, advance, accelerate, decelerate, etc. The navigation algorithm is what we have described as a large conceptual space loop. Once the execution of this loop has been completed, time is incremented. We compare the predefined destination (exit ramp) of each vehicle with a neighbourhood of the cell where the vehicle is currently located. Those vehicles that have reached their exit ramp are removed from the expressway. Exit ramps are listed in the input configuration file without any other parameter than a keyword and the number corresponding to the cell where the exit ramp is located.
Information about all vehicles is logged to an output data file. This information is not aggregate information, but it is individual information about the location of each vehicle at the end of the execution of each time step. This output file allows calculating, off line, at the end of the simulation, the exact travel time of each vehicle, from entry ramp to exit ramp. The average of all the individual travel times is the travel time as earlier defined. Aggregate information is output to a different data file where we store: current time step number, total number of vehicles instantiated, total number of vehicles on the road, and total number of vehicles delayed in entry ramps. Thus, it is possible to infer how many vehicles have exited at this time.
When the maximum simulation time, as defined in the input configuration file, is reached, some housekeeping work is carried on and the execution is terminated. The navigation algorithm is divided into the following subsets:

- Change lane to the right if required (e.g., if no vehicle must be passed, as required by the rules of traffic, or if the vehicle is approaching its exit ramp);
- Change lane to the left if required (e.g., if a slower vehicle has to be passed or an obstacle has to be avoided);
- Advance, either at constant speed, if travelling at maximum (vehicle specific) speed, or accelerating/decelerating as it may be required by the traffic situation;
- Randomly, as specified by a command in the input configuration file, according to a predefined probability, execute an erratic behaviour if required.

For each of the above subsets, lane number and cell number are initialized to zero. Two buffers (i.e., arrays) are setup: OldBuffer is set up to contain a snapshot of the current traffic situation, with the location of each vehicle; NewBuffer is empty. For each lane, all cells are examined individually. If the cell is empty, i.e. there is no vehicle at that cell, nothing happens. If a vehicle is located in the cell under consideration, the algorithm required by the subset being executed is applied. All algorithms are of CA (Cellular Automata) like algorithms and are applied to the cell and a neighbourhood, i.e. a number of cells around it. Each lane is treated as a CA. Changing lane is logically equivalent to jumping from one CA to another one. However, the actual implementation uses 1-D single GCA. When all cells have been scanned and the related processing has been done, NewBuffer is copied into OldBuffer and the display is refreshed if the model is being executed in graphic mode.
Modeling multilane highway traffic with CA introduces some potential conflict whenever more than one vehicle “wants to move”to the same cell. This is not different from what happens in real life when, for example, a car is arriving at high speed on the leftmost lane and another car is changing lane from the centre lane to the leftmost lane. In real life, drivers can change their actions because time is continuous

and because decision making is continuous and instantaneous. In a CA model, because all decisions are made based on the state of the CA at time t-1 and implemented at time t, we can have a conflict. In a traditional ECA, no vehicle can move ahead of another vehicle, so there is no conflict. In a 1-D CA, even if higher speeds are modelled, no vehicle can move ahead of another vehicle, so there is no conflict. In a 2-D CA, there is a potential conflict.

The "Global Cellular Automata with Write access"(GCA-w) developed by Rolf Hoffmann and his collaborators allows solving the potential conflicts, [15], [16], [17], [18], and [19]. In the GCA-w model each occupied cell can have write access to the neighbours and can update its neighbours' private member variable. Thus, before moving, a vehicle can issue a signal to the other vehicles in potential conflict and give them an early warning of its intention of moving to a given cell. This is simply done by setting a flag in a private member variable of the other vehicle.

We have decided to assign the value of 3 seconds to each time step. Thus, the minimum speed of a vehicle advancing by one cell at each time step is equivalent to 9 km/h (that is, 7.5 x 3600/3 = 7.5 x 1200 = 9000 m/h). This allows representing most realistic and legal speeds observed in Canadian expressways, with a vehicle advancing by a maximum of 11 cells per time step, that is, 99 km/h, as the speed limit is at 100 km/h. This is different from the model of Nagel and Schreckenberg, which uses 1 second per time step. We are currently comparing the results of our model with realistic traffic data in Ontario to decide if our choice is appropriate or needs to be revisited.

Details about our "Multilane Single GCA-w Agent Based Model"can be found in [20].

3.2 Example of Digital Experiment

As an example of the type of digital experiment that can be conducted with the software package implementing our highway traffic model, we show how it can be used to verify the impact of the relative position of entry and exit ramps on travel time.

We consider two topologies: "Expressway with Entry Ramp Preceding Exit Ramp"as in Figure 1 and "Expressway with Exit Ramp Preceding Entry Ramp"as in Figure 2.

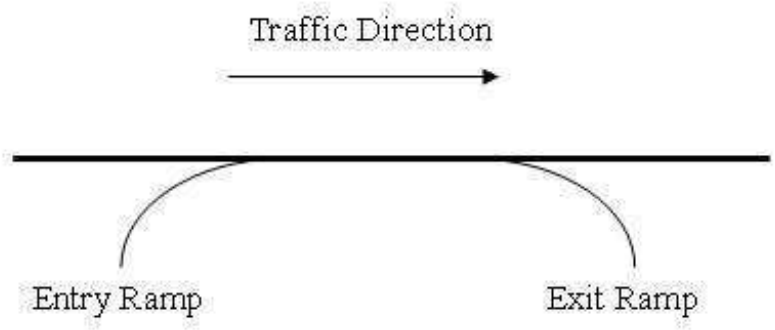


Fig. 1: Expressway with Entry Ramp Preceding Exit Ramp

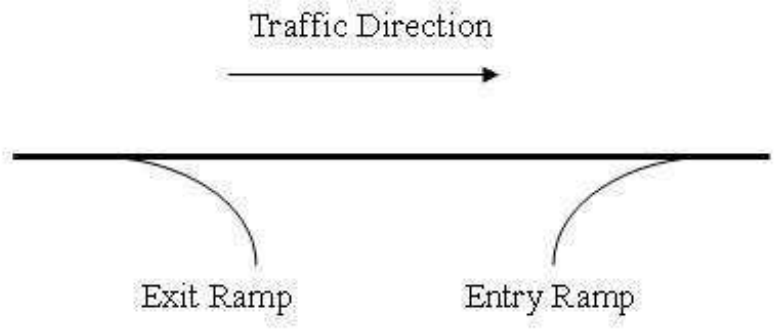


Fig. 2: Expressway with Exit Ramp Preceding Entry Ramp

The actual choice of one topology or the other depends on many factors, such as if the intersecting road is above or below the expressway. For the purpose of this example we ignore such factors and we focus only on the impact of the two topologies on travel time.

We assume that the stretch of expressway under consideration is 30 km long and, thus, consists of 4000 cells (i.e. 3000 / 7.5 = 4000). We consider only one direction of travel, rightward (i.e. west - east). We assume that for Figure 1 the entry ramp is located at cell 2000 (km 15 from the beginning of the expressway under consideration) and that the exit ramp is located at cell 2200 (km 16.5 from the beginning of the expressway. This is an approximation for the sake of simplicity in this example. In reality, entry and exit ramps involve more than one cell. In both cases the rightmost lane and the ramp run parallel for about 20 cells. When the entering or exiting vehicle is up to the proper speed or down to proper speed, it will leave the ramp or move to the ramp as it may apply. Likewise in Figure 2 the exit ramp is located at cell 2000 (km 15 from the beginning of the expressway under consideration) and the entry ramp is located at cell 2200 (km 16.5 from the beginning of the expressway. As in Figure 1 in reality, entry and exit ramps involve more than one cell.

Table 1 shows the travel time, from entry ramp to exit ramp, given a certain vehicle creation probability (shown in the leftmost column) and given a certain probability that the created vehicle will head to cell number 4000. For each couple of probabilities, vehicle creation probability and probability that the created vehicle we head to cell 4000, we show also total number of cars. As expected this number increases with the vehicle creation probability. From this experiment not much can be said about the impact of the probability that the created vehicle will head to cell 4000.

Table 2 is constructed similarly to Table 1. It makes sense to compare only the travel time of the cars travelling from cell number 0 to cell 4000 (actually cell number 3999, because the other travel segments are of different length in Table 1 and Table 2. The travel time of these cars is consistently shorter in the experiment of Table 2. Presumably this is due to the fact that the topology with “Exit Ramp Preceding Entry Ramp”removes cars from the expressway before new cars are introduced, thus locally reducing the density of cars. Also in the “Entry Ramp Preceding Exit Ramp”topology and car moving from lane 1 to lane 0 to exit may slow down a car that just entered the expressway, thus slowing down also other cars.

	Probability of Heading To Cell 4000 = 0.3	Probability of Heading To Cell 4000 = 0.5
Creation Probability at Cell 0 = 0.3	0⟶2200; 202 time steps 0⟶3999; 365 time steps 2000 ⟶ 3999; 184 time steps Total # of Cars: 1678	0 ⟶ 2200; 201 time steps 0⟶3999; 365 time time setps 2000⟶ 3999; 184 time steps Total # of Cars: 1674
Creation Probability at Cell 0 = 0.4	0⟶2200; 202 time steps 0⟶3999; 366 time steps 2000 ⟶ 3999; 184 time steps Total# of Cars: 1996	0 ⟶ 2200; 202 time steps 0⟶3999; 366 time steps 2000⟶ 3999; 184 time steps Total # of Cars: 1986
Creation Probability at Cell 0 = 0.5	0⟶2200; 204 time steps 0⟶3999; 369 time steps 2000 ⟶ 3999; 186 time steps Total # of Cars: 2349	0 ⟶ 2200; 205 time steps 0⟶3999; 369 time steps 2000⟶ 3999; 187 time steps Total # of Cars: 2339

Tab. 1: Average Travel Time for Expressway with Entry Ramp Preceding Exit Ramp - Simulation Time = 3600 Time Steps

	Probability of Heading To Cell 4000 = 0.3	Probability of Heading To Cell 4000 = 0.5
Creation Probability at Cell 0 = 0.3	0⟶2200; 182 time steps 0⟶3999; 364 time steps 2000 ⟶ 3999; 164 time steps Total # of Cars: 1691	0 ⟶ 2200; 182 time steps 0⟶3999; 364 time time steps 2000⟶ 3999; 164 time steps Total # of Cars: 1684
Creation Probability at Cell 0 = 0.4	0⟶2200; 182 time steps 0⟶3999; 364 time steps 2000 ⟶ 3999; 164 time steps Total# of Cars: 2005	0 ⟶ 2200; 182 time steps 0⟶3999; 364 time steps 2000⟶ 3999; 164 time steps Total # of Cars: 1994
Creation Probability at Cell 0 = 0.5	0⟶2200; 182 time steps 0⟶3999; 364 time steps 2000 ⟶ 3999; 164 time steps Total # of Cars: 2363	0 ⟶ 2200; 182 time steps 0⟶3999; 364 time steps 2000⟶ 3999; 164 time steps Total # of Cars: 2347

Tab. 2: Average Travel Time for Expressway with Exit Ramp Preceding Entry Ramp - Simulation Time = 3600 Time Steps.

We would like to emphasize that this experiment has been presented only as an example of the capability of the model and of the resulting digital laboratory. This is not a complete study of the effect of topology of the relative position of entry & exit ramps over travel time. A much larger number of experiments are required to reach conclusions, accounting for different expressway topologies (i.e., number of entry and exit ramps), different traffic densities (i.e. vehicle creation probability at each ramp). Also the length of the expressway under consideration is expected to be a factor. We are actually conducting these and other experiments and we plan to discuss them elsewhere.

4 Future Work

We are currently validating our model with information from traffic engineers. We plan on using this model for practical traffic engineering applications, to estimate how some technological innovations affects travel time between two access ramps, an entry ramp and an exit ramp, once certain highway traffic parameters are known at certain points of the highway. Our concern is primarily with effects of flow and congestion through a long highway on travel time. The technological innovations include wireless communication from roadside transmitters to vehicles, wireless communication among vehicles, etc. We are also investigating ways of reducing the computational overhead of handling entry and exit ramps. We are considering parallelizing our code for execution under SHARCNET, a consortium of Canadian academic institutions sharing a network of high performance computers, see [21].

Acknowledgements

A.T. Lawniczak acknowledges partial financial support from the Natural Science and Engineering Research Council (NSERC) of Canada. B.N. Di Stefano acknowledges full financial support from Nuptek Systems Ltd. A.T. Lawniczak acknowledges support from SHARCNET in the form of computing facilities and resources for the Linux implementation and testing of the software. The authors thank The Fields Institute for Research in Mathematical Sciences for providing hospitality and Prof. Danuta Makowiec and Prof. Rolf Hoffmann for providing inspiring conversation.

References

[1] Transportation Engineering - Online Lab Manual, © 2000, 2001, 2002, 2003, Oregon State University, Portland State University, University of Idaho, Glossary
`http://www.webs1.uidaho.edu/niatt_labmanual/Chapters/TrafficFlowTheory/Glossary/index.htm`

[2] M.J. Lighthill, G.B. Whitham, Proc. R. Soc. A229, 317 (1955).

[3] P.I. Richards, Operations Research 4, 42 (1956).

[4] M. Cremer, J. Ludwig, Mathematical and Computers in Simulation 28, 297 (1986).

[5] Nagel K., Schreckenberg M. (1992). A cellular automaton model for freeway traffic. J. Physique I 2, 2221 - 2229.

[6] W. Knospe, L. Santen, A. Schadschneider, M. Schreckenberg, Phys. Rev. E70, 016115 (2004).

[7] W. Knospe, L. Santen, A. Schadschneider, M. Schreckenberg, J. Phys. A33, L477 (2000).

[8] W. Knospe, L. Santen, A. Schadschneider, M. Schreckenberg, Phys. Rev. E65, 056133 (2002).

[9] M. Schreckenberg, A. Schadschneider, K. Nagel, N. Ito, Phys. Rev. E51, 2939 (1995).

[10] P. Wagner, K. Nagel, D. Wolf, Physica A 234, 687 (1997).

[11] D. Chowdhury, L. Santen, A. Schadschneider, Phys. Rep. 329, 199 (2000).

[12] D. Helbing, Rev. Mod. Phys 73, 1067 (2001).

[13] Anna T. Lawniczak and Bruno N. Di Stefano, Development of CA model of highway traffic, in Adamatzky A., Alonso-Sanz R., Lawniczak A., Martinez G. J., Morita K., Worsch T. (Editors), Automata-2008. Theory and Applications of Cellular Automata. (Luniver Press, 2008), 14 pages

[14] Maerivoet S. and De Moor B.(2005). Cellular Automata Models of Road Traffic, in Physics Reports, vol. 419, nr. 1, pages 1-64, November 2005.

[15] Hoffmann, R., Volkmann, K.-P., Waldschmidt, S.: Global Cellular Automata GCA: An Universal Extension of the CA Model. In: Worsch, Thomas (Editor): ACRI Conference (2000).

[16] Hoffmann, R., Volkmann, K.-P., Waldschmidt, S., Heenes, W.: GCA: Global Cellular Automata, A Flexible Parallel Model. In Proceedings of: 6th International Conference on Parallel Computing Technologies PaCT 2001, Lecture Notes in Computer Science (LNCS 2127), Springer (2001).

[17] Hoffmann, R., Volkmann, K.-P., Heenes, W.: GCA: A massively parallel Model. IPDPS 2003, IEEE Comp. Soc.

[18] Ehrt, Chr.: Globaler Zellularautomat: Parallele Algorithmen. Diplomarbeit,Technische Universitat Darmstadt, 2005. http://www.ra.informatik.tudarmstadt.de/forschung/publikationen/.

[19] Heenes, W., Hoffmann, R., Jendrsczok, J.: A Multiprocessor Architecture for the Massively Parallel Model GCA. IPDPS/SMTPS 2006, IEEE Proceedings: 20th International Parallel & Distributed Processing Symposium.

[20] A.T. Lawniczak, B.N. Di Stefano. Digital Laboratory of Agent-based Highway Traffic Model, Acta Physica Polonica B Proceedings Supplement Vol. 3, No. 2, February 2010, pp 479-453.

[21] https://www.sharcnet.ca/

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 255–264

Composition, Union and Division of Cellular Automata on Groups

Takahiro Ito[1] and Mitsuhiko Fujio[2] and Shuichi Inokuchi[3] and Yoshihiro Mizoguchi[3]

[1] *Graduate School of Mathematics, Kyushu University, JAPAN*
[2] *Department of Systems Design and Informatics, Kyushu Institute of Technology, JAPAN*
[3] *Faculty of Mathemaics, Kyushu University, JAPAN*

We introduce the notion of 'Composition', 'Union' and 'Division' of cellular automata on groups. A kind of notions of compositions was investigated by Sato (1994) and Manzini (1998) for linear cellular automata, we extend the notion to general cellular automata on groups and investigated their properties. We observe the all unions and compositions generated by one-dimensional 2-neighborhood cellular automata over $\mathbf{Z}_2$ including non-linear cellular automata. Next we prove that the composition is right-distributive over union, but is not left-distributive. Finally, we conclude by showing reformulation of our definition of cellular automata on group which admit more than three states. We also show our formulation contains the representation using formal power series for linear cellular automata in Manzini (1998).

Keywords: Cellular automata, Groups, Models of computation, Automata

1 Introduction

The study of cellular automata was initiated by von Neumann (1983) and have been developed by many researchers as a good computational model for physical systems simulation. Recently cellular automata have been investigated in various fields including computer science, biology, physics, since they provide simple and powerful models for parallel computation and natural phenomena.

In this paper, we investigate cellular automata on groups as a formal model of computation. To compose simple cellular automata into a complex cellular automaton, we introduce the notion of 'Composition' of cellular automata on groups. The notion of automata on groups was first treated as a special case for automata on graphs (Caley graphs) which represent groups in Róka (1994); Rémila (1998). Watanabe and Noguchi (1982) investigated the decomposition of finite automata from the view point of spatial networks using groups. Pries et al. (1986) investigated cellular automata as a tool for implementing hardware algorithms in VLSI. They considered configurations decided by a cellular automaton as a group and divided configurations into simple configurations using group properties. Sato (1994) introduced group structured linear cellular automata and the star operation of local transition rules. The star operation is a kind of composition of cellular automata but the definition of it is different from ours. Manzini (1998) also

investigated the linear cellular automata using the formal power series and their product to find inverse local transition functions. The product of formal power series are equal to our composition of cellular automata for linear cases. An abstract collision system in Ito et al. (2008) is considered as an extension of a cellular automaton, the notion of 'composition' for an abstract collision system on G-sets is investigated in Ito (2010).

This paper follows on from Fujio (2008). He introduced the composition of cellular automata on groups in order to reduce a complex behaved dynamics into simpler ones. We introduce a formal definition of cellular automata on group over $\mathbf{Z}_2$. In our framework, operations on cellular automata 'Union', 'Division' and 'Composition' are introduced. Unions of all 2-neighborhood cellular automata are investigated. Compositions of all 2-neighborhood cellular automata are also investigated and determined the subset of 3-neighborhood cellular automata which are generated by composing two 2-neighborhood cellular automata. Next we prove that the composition is right-distributive over union, but is not left-distributive. Finally, we conclude by showing reformulation of our definition of cellular automata on group which admit more than three states. We also show our formulation contains the representation using formal power series for linear cellular automata in Manzini (1998).

2 Cellular Automata on Groups

Definition 1 *Let G be a group. A cellular automaton on C is a triple $C = (G, V, V')$ of a group G, subsets $V \subset G$ and $V' \subset 2^V$. For V', we define functions $l_{V'} : 2^V \to \{\phi, \{e\}\}$ by*

$$l_{V'}(X) = \begin{cases} \phi & (X \notin V') \\ \{e\} & (X \in V'), \end{cases}$$

and $F_C : 2^G \to 2^G$ by $F_C(\mathbf{c}) = \bigcup_{g \in G} g l_{V'}(g^{-1}\mathbf{c} \cap V)$. We call the map $l_{V'}$ a local transition function and F_C a global transition function.

Proposition 2 *Let $C_1 = (G, V_1, V_1')$ and $C_2 = (G, V_2, V_2')$ be cellular automata. If*

$$e \in F_{C_1}(\mathbf{c}) \;\Leftrightarrow\; e \in F_{C_2}(\mathbf{c}) \;\; \textit{(for any } \mathbf{c} \in 2^G\textit{)}$$

then $F_{C_1} = F_{C_2}$

Proof. Since $F_{C_1}(\mathbf{c}) = \{g \in G \,|\, l_{V_1'}(g^{-1}\mathbf{c} \cap V_1) = \{e\}\} = \{g \in G \,|\, g^{-1}\mathbf{c} \cap V_1 \in V_1'\}$, we have $g \in F_{C_1}(\mathbf{c}) \Leftrightarrow g^{-1}\mathbf{c} \cap V_1 \in V_1' \Leftrightarrow e \in F_{C_1}(g^{-1}\mathbf{c}) \Leftrightarrow e \in F_{C_2}(g^{-1}\mathbf{c}) \Leftrightarrow g^{-1}\mathbf{c} \cap V_2 \in V_2' \Leftrightarrow g \in F_{C_2}(\mathbf{c})$. □

In the followings, we consider the set of all integers $\mathbf{Z}$ as an additive group $\mathbf{Z} = (\mathbf{Z}, +, 0)$. So usual one dimensional cellular automata with 2-states are represented as cellular automata on the group $\mathbf{Z}$. We define 2-neighborhood and 3-neighborhood 2-states cellular automata in the next definition.

Definition 3 *For $k \geq 1$ and $n \in \{0, 1, \cdots, 2^{2^k} - 1\}$, we define cellular automata $CA(k, n)$ on $\mathbf{Z}$ by $CA(k, n) = (\mathbf{Z}, V, V_n')$ where $V = \{0, 1, \cdots, k-1\}$, and V_n' is the subset of 2^V which satisfies $n = \sum_{X \in V_n'} 2^{\sum_{i \in X} 2^i}$.*

We note $CA(1, 0) = (\mathbf{Z}, \{0\}, \phi)$ and $CA(1, 1) = (\mathbf{Z}, \{0\}, \{0\})$.

Example 4 *Since* $6 = 2+2^2 = 2^{2^0}+2^{2^1}$*, we have* $CA(2,6) = (\mathbf{Z}, \{0,1\}, \{\{0\},\{1\}\})$*. Since* $90 = 2+2^3+2^4+2^6 = 2^{2^0}+2^{2^0+2^1}+2^{2^2}+2^{2^1+2^2}$*, we have* $CA(3,90) = (\mathbf{Z}, \{0,1,2\}, \{\{0\},\{2\},\{0,1\},\{1,2\}\})$*. The elements* X *in* V'_n *represents the state of neighborhood which induce the next states '1'. For a rule number 90, we have the following table:*

Neighborhood	*111*	*110*	*101*	*100*	*011*	*010*	*001*	*000*
$X \in V'_n$	$\{0,1,2\}$	$\{1,2\}$	$\{0,2\}$	$\{2\}$	$\{0,1\}$	$\{1\}$	$\{0\}$	ϕ
$l'_V(X)$	ϕ	$\{e\}$	ϕ	$\{e\}$	$\{e\}$	ϕ	$\{e\}$	ϕ

The configuration $\mathbf{c} \subset \mathbf{Z}$ represents places where the state is 1. Since $n \in F_C(\mathbf{c}) \Leftrightarrow l_{V'}(n^{-1}\mathbf{c} \cap V) = \{e\} \Leftrightarrow n^{-1}\mathbf{c} \cap V \in V' \Leftrightarrow \mathbf{c} \cap nV \in nV'$, the next state at n is 1 if $\mathbf{c} \cap nV \in nV'$. For 3-neighborhood case we are choosing $V = \{0,1,2\}$, the left-hand side of the state is changing. It seems to be better that we choose $V = \{-1,0,1\}$ but it is not convenient for even-neighborhood case. Our numbered 3-neighborhood cellular automata $CA(3,n)$ is a shifted version of usual numbered elementary cellular automata. Later, we define a cellular automaton SHIFT which represent a shift operation and a operator 'composition' ($\diamond$) of two cellular automata. After that the usual numbered elementary cellular automata is represented as SHIFT$\diamond CA(3,n)$.

Example 5 *SHIFT* $= (\mathbf{Z}, \{-1,0\}, \{\{-1\},\{-1,0\}\})$ *is a cellular automata on group* $\mathbf{Z}$*.*

$\mathbf{Z}^2$ is also considered as a group, so it is easy to represent a multi-dimensional cellular automata such as The Game of Life (Berlekamp et al. (1982)) as a cellular automata on a group.

Example 6 *LIFE* $= (\mathbf{Z}^2, V_{LIFE}, V'_{LIFE})$ *is a cellular automata on group* $\mathbf{Z}^2$*, where*

$$V_{LIFE} = \{\begin{pmatrix}-1\\-1\end{pmatrix}, \begin{pmatrix}0\\-1\end{pmatrix}, \begin{pmatrix}+1\\-1\end{pmatrix}, \begin{pmatrix}-1\\0\end{pmatrix}, \begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}+1\\0\end{pmatrix}, \begin{pmatrix}-1\\+1\end{pmatrix}, \begin{pmatrix}0\\+1\end{pmatrix}, \begin{pmatrix}+1\\+1\end{pmatrix}\},\ \textit{and}$$

$$V'_{LIFE} = \{v \in 2^V \mid (\#v = 3) \vee (\#v = 4 \wedge \begin{pmatrix}0\\0\end{pmatrix} \in v)\}.$$

We note that $\#v$ *is the number of elements in a set* v*.*

One dimensional cellular automaton on $\mathbf{Z}$ is embedded into the two dimensional cellular automaton on $\mathbf{Z}^2$. We define two natural embeddings EX and EY in the following.

Definition 7 *For a cellular automata* $C = (\mathbf{Z}, V, V')$*, we define a cellular automata* $EX(C)$ *on* $\mathbf{Z}^2$ *by* $EX(C) = (\mathbf{Z}^2, V_{EX(C)}, V'_{EX(CA)})$ *where*

$$V_{EX(C)} = \{\begin{pmatrix}x\\0\end{pmatrix} \mid x \in V\},\ \textit{and}$$

$$V'_{EX(C)} = \{\{\begin{pmatrix}x\\0\end{pmatrix} \mid x \in X\} \mid X \in V'\}.$$

We also define a cellular automata $EY(C)$ *on* $\mathbf{Z}^2$ *by* $EY(C) = (\mathbf{Z}^2, V_{EY(C)}), V'_{EY(C)})$ *where*

$$V_{EY(C)} = \{\begin{pmatrix}0\\x\end{pmatrix} \mid x \in V\},\ \textit{and}$$

$$V'_{EY(C)} = \{\{\begin{pmatrix}0\\x\end{pmatrix} \mid x \in X\} \mid X \in V'\}.$$

$n\backslash m$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	3	3	5	5	7	7	9	9	11	11	13	13	15	15
2	2	3	2	3	6	7	6	7	10	11	10	11	14	15	14	15
3	3	3	3	3	7	7	7	7	11	11	11	11	15	15	15	15
4	4	5	6	7	4	5	6	7	12	13	14	15	12	13	14	15
5	5	5	7	7	5	5	7	7	13	13	15	15	13	13	15	15
6	6	7	6	7	6	7	6	7	14	15	14	15	14	15	14	15
7	7	7	7	7	7	7	7	7	15	15	15	15	15	15	15	15
8	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
9	9	9	11	11	13	13	15	15	9	9	11	11	13	13	15	15
10	10	11	10	11	14	15	14	15	10	11	10	11	14	15	14	15
11	11	11	11	11	15	15	15	15	11	11	11	11	15	15	15	15
12	12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
13	13	13	15	15	13	13	15	15	13	13	15	15	13	13	15	15
14	14	15	14	15	14	15	14	15	14	15	14	15	14	15	14	15
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

Fig. 1: Table of unions : $CA(2,n) \cup CA(2,m)$

Definition 8 *Let $1 \leq k < k'$, $0 \leq x \leq k' - k$ and $CA(k,n) = (\mathbf{Z}, V, V')$. $CA(k,n)^{k'}_x$ is defined by $CA(k,n)^{k'}_x = (\mathbf{Z}, \{0,1,\cdots,k'-1\}, V'')$ where*

$$\begin{aligned} V'' &= \{s_1 \cup v \cup s_2 \,|\, s_1 \in S_1, s_2 \in S_2, v \in V'\}, \\ S_1 &= \begin{cases} \{\phi\} & (x=0) \\ 2^{\{0,\cdots,x-1\}} & (x>0) \end{cases}, \\ S_2 &= 2^{\{x+1,\cdots,k\}} \end{aligned}$$

We note that $F_{CA(k,n)^{k'}_0} = F_{CA(k,n)}$ and $F_{CA(k,n)^{k'}_1} = SHIFT \diamond F_{CA(k,n)}$.

Definition 9 (Union) *Let $C_1 = (G, V_1, V_1')$ and $C_2 = (G, V_2, V_2')$ be cellular automata on G. The union $C_1 \cup C_2$ of C_1 and C_2 is defined by $C_1 \cup C_2 = (G, V_1 \cup V_2, V_1' \cup V_2')$.*

Definition 10 (Division) *Let $C = (G, V, V')$ be a cellular automaton on G. If there exist $C_1 = (G, V_1, V_1')$ and $C_2 = (G, V_2, V_2')$ be cellular automata on G such that $V = V_1 \cup V_2$ and $V' = V_1' \cup V_2'$, then we call C_1 and C_2 are division of C and C is dividable by C_1 and C_2.*

Example 11 *The class of all 2-neighborhood cellular automata $\{CA(2,n) \,|\, n = 0,..,15\}$ is generated by $\{CA(2,0)$, $CA(2,1)$, $CA(2,2)$, $CA(2,4)$, $CA(2,8)\}$ using 'union' operations. For example, $CA(2,13)$ is dividable by $CA(2,1)$, $CA(2,4)$, and $CA(2,8)$. Fig 1 is the table of unions for $CA(2,n)$ ($n = 0,..,15$).*

Definition 12 (Composition) *Let $C_1 = (G, V_1, V_1')$ and $C_2 = (G, V_2, V_2')$ be cellular automata on G. The composition $C_1 \diamond C_2$ of C_1 and C_2 is defined by $C_1 \diamond C_2 = (G, V_1 \cdot V_1, V_1' \diamond V_2')$ where*

$$\begin{aligned} V_1 \cdot V_2 &= \{v_1 v_2 \in G \,|\, v_1 \in V_1, v_2 \in V_2\} \text{ and} \\ V_1' \diamond V_2' &= \{X \in 2^{V_1 \cdot V_2} \,|\, \{v \in V_1 \,|\, v^{-1}X \cap V_2 \in V_2'\} \in V_1'\}. \end{aligned}$$

Example 13 *Let $C = (\mathbf{Z}, V, V')$ be a cellular automaton on $\mathbf{Z}$ where $V = \{0,1\}$ and $V' = \{\{0\},\{1\}\}$. We have $V \cdot V = \{0,1,2\}$. Since $1^{-1}\{0,1\} \cap \{0,1\}$ $\{0-1, 1-1\} \cap \{0,1\} = \{0\}$ and $0^{-1}\{1,2\} \cap \{0,1\} = \{1-0, 2-0\} \cap \{0,1\} = \{1\}$, we have $V' \diamond V' = \{\{0\},\{2\},\{0,1\},\{1,2\}\}$. So we have $CA(2,6) \diamond CA(2,6) = CA(3,90)$.*

$n\backslash m$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	255	236	209	192	139	136	129	128	55	36	17	0	3	0	1	0
2	0	16	34	48	68	68	66	64	8	24	34	48	12	12	2	0
3	255	252	243	240	207	204	195	192	63	60	51	48	15	12	3	0
4	0	2	12	12	48	34	24	8	64	66	68	68	48	34	16	0
5	255	238	221	204	187	170	153	136	119	102	85	68	51	34	17	0
6	0	18	46	60	116	102	90	72	72	90	102	116	60	46	18	0
7	255	254	255	252	255	238	219	200	127	126	119	116	63	46	19	0
8	0	1	0	3	0	17	36	55	128	129	136	139	192	209	236	255
9	255	237	209	195	139	153	165	183	183	165	153	139	195	209	237	255
10	0	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255
11	255	253	243	243	207	221	231	247	191	189	187	187	207	221	239	255
12	0	3	12	15	48	51	60	63	192	195	204	207	240	243	252	255
13	255	239	221	207	187	187	189	191	247	231	221	207	243	243	253	255
14	0	19	46	63	116	119	126	127	200	219	238	255	252	255	254	255
15	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Fig. 2: Table of compositions : $CA(2,n)\diamond CA(2,m)$

Example 14 *The rule numbers of the 3-neighborhood cellular automata generated by composing 2-neighborhood cellular automata is* $\{0, 1, 2, 3, 8, 12, 15, 16, 17, 18, 19, 24, 34, 36, 46, 48, 51, 55, 60, 63, 64, 66, 68, 72, 85, 90, 102, 116, 119, 126, 127, 128, 129, 136, 139, 153, 165, 170, 183, 187, 189, 191, 192, 195, 200, 204, 207, 209, 219, 221, 231, 236, 237, 238, 239, 240, 243, 247, 252, 253, 254, 255\}$. *There are 62 kinds of 3-neighborhood cellular automata. Fig 2 is the table of compositions for* $CA(2,n)$ *(*$n = 0, .., 15$*).*

Lemma 15 *Let* $C = (G, V, V')$ *be a cellular automaton and* $V_0 \subset G$*. For any* $\mathbf{c} \in 2^G$*,*

$$F_C(\mathbf{c}) \cap V_0 = F_C(\mathbf{c} \cap (V_0 \cdot V)) \cap V_0$$

Proof. We have $F_C(\mathbf{c}) \cap V_0 = \{v_0 \in V_0 \mid v_0^{-1}\mathbf{c} \cap V \in V'\} = \{v_0 \in V_0 \mid \mathbf{c} \cap v_0 V \in v_0 V'\} = \{v_0 \in V_0 \mid (\mathbf{c} \cap V_0 \cdot V) \cap v_0 V \in v_0 V'\} = F_C(\mathbf{c} \cap (V_0 \cdot V)) \cap V_0$. □

The composition of cellular automata corresponds to find a cellular automaton which global transition function is the composition of global transition functions of original cellular automata.

Theorem 16 (Fujio (2008))

$$F_{C_1} \circ F_{C_2} = F_{C_1 \diamond C_2}$$

Proof. Since $F_{C_2}(\mathbf{c}) \cap V_1 = \{v \in V_1 \mid v^{-1}\mathbf{c} \cap V_2 \in V_2'\}$, we have

$$\begin{aligned}
e \in F_{C_1}(F_{C_2}(\mathbf{c})) &\Leftrightarrow F_{C_2}(\mathbf{c}) \cap V_1 \in V_1' \\
&\Leftrightarrow F_{C_2}(\mathbf{c} \cap V_1 \cdot V_2) \cap V_1 \in V_1' \text{ (by Proposition. 2)} \\
&\Leftrightarrow \{v_1 \in V_1 \mid v^{-1}(\mathbf{c} \cap V_1 \cdot V_2) \cap V_2 \in V_2'\} \in V_1' \\
&\Leftrightarrow \mathbf{c} \cap V_1 \cdot V_2 \in V_1' \diamond V_2' \\
&\Leftrightarrow e \in F_{C_1 \diamond C_2}(\mathbf{c})
\end{aligned}$$

□

Theorem 17 *Let* $C_1 = (G, V, V_1')$*,* $C_2 = (G, V, V_2')$ *and* $C_3 = (G, V_3, V_3')$ *be cellular automata on a group* G*. Then,*

$$(C_1 \cup C_2)\diamond C_3 = (C_1 \diamond C_3) \cup (C_2 \diamond C_3)$$

Proof. First, we note

$$\begin{aligned}(C_1 \cup C_2)\diamond C_3 &= (G, V\cdot V_3, (V_1' \cup V_2')\diamond V_3') \text{ , and}\\ (C_1\diamond C_3)\cup(C_2\diamond C_3) &= (G, V\cdot V_3, (V_2'\diamond V_1')\cup(V_3'\diamond V_1')).\end{aligned}$$

Next, we have

$$\begin{aligned}(V_1' \cup V_2')\diamond V_3' &= \{X\in 2^{V\cdot V_3} \mid \{v\in V \mid v^{-1}X\cap V_3 \in V_3'\}\in V_1'\cup V_2'\}\\ &= \{X\in 2^{V\cdot V_3} \mid \{v\in V \mid v^{-1}X\cap V_3 \in V_3'\}\in V_1'\}\\ &\quad \cup\{X\in 2^{V\cdot V_3} \mid \{v\in V \mid v^{-1}X\cap V_3 \in V_3'\}\in V_2'\}\\ &= (V_1'\diamond V_3')\cup(V_2'\diamond V_3')\end{aligned}$$

□

We note that $C_1\diamond(C_2\cup C_3) = (C_1\diamond C_2)\cup(C_1\diamond C_3)$ does not always holds for cellular automata C_1, C_2 and C_3. For example $CA(2,6)\diamond(CA(2,2)\cup CA(2,4)) = CA(2,6)\diamond CA(2,6) = CA(3,90)$, and $(CA(2,6)\diamond CA(2,2))\cup(CA(2,6)\diamond CA(2,4)) = CA(3,46)\cup CA(3,116) = CA(3,126)$.

Proposition 18 *Let $CA(1,n)_x^{k_1}$, $CA(k_2,n_2)$ and $CA(k_2,n_3)$ be cellular automata on $\mathbf{Z}$, where $0\le x< k_1$, and $n=0,1$. Then,*

$$CA(1,n)_x^{k_1}\diamond(CA(k_2,n_2)\cup CA(k_2,n_3)) = (CA(1,n)_x^{k_1}\diamond CA(k_2,n_2))\cup(CA(1,n)_x^{k_1}\diamond CA(k_2,n_3)).$$

Proof. Let $V_1=\{0,\cdots,k_1-1\}$, $V_1'=\{X\in 2^V \mid x\in X\})$, $\bar{V_1'}=\{X\in 2^V \mid x\notin X\})$, $CA(k_2,n_2) = (\mathbf{Z},V_2,V_2')$, and $CA(k_2,n_3)=(\mathbf{Z},V_2,V_3')$. First, we note

$$\begin{aligned}CA(1,0)_x^{k_1} &= (\mathbf{Z},V_1,\bar{V_1'}),\\ CA(1,1)_x^{k_1} &= (\mathbf{Z},V_1,V_1'),\\ CA(1,0)_x^{k_1}\diamond(CA(k_2,n_2)\cup CA(k_2,n_3)) &= (\mathbf{Z},V_1\cdot V_2, V_1'\diamond(V_2'\cup V_3')), \text{and}\\ (CA(1,n)_x^{k_1}\diamond CA(k_2,n_2))\cup(CA(1,n)_x^{k_1}\diamond CA(k_2,n_3)) &= (\mathbf{Z},V_1\cdot V_2,(V_1'\diamond V_2')\cup(V_1'\diamond V_3')).\end{aligned}$$

Since

$$\begin{aligned}V_1'\diamond(V_2'\cup V_3') &= \{X\in 2^{V_1\cdot V_2} \mid \{v\in V \mid v^{-1}X\cap V_2\in(V_2'\cup V_3')\}\in V_1'\}\\ &= \{X\in 2^{V_1\cdot V_2} \mid x^{-1}X\cap V_2\in(V_2'\cup V_3')\}, \text{and}\\ (V_1'\diamond V_2')\cup(V_1'\diamond V_3') &= \{X\in 2^{V_1\cdot V_2} \mid \{v\in V \mid v^{-1}X\cap V_2\in V_2'\}\in V_1'\}\\ &\quad \cup\{X\in 2^{V_1\cdot V_2} \mid \{v\in V \mid v^{-1}X\cap V_2\in V_3'\}\in V_1'\}\\ &= \{X\in 2^{V_1\cdot V_2} \mid x^{-1}X\cap V_2\in V_2'\}\\ &\quad \cup\{X\in 2^{V_1\cdot V_2} \mid x^{-1}X\cap V_2\in V_3'\},\end{aligned}$$

we have $V_1'\diamond(V_2'\cup V_3') = (V_1'\diamond V_2')\cup(V_1'\diamond V_3')$, and

$$CA(1,1)_x^{k_1}\diamond(CA(k_2,n_2)\cup CA(k_2,n_3)) = (CA(1,1)_x^{k_1}\diamond CA(k_2,n_2))\cup(CA(1,1)_x^{k_1}\diamond CA(k_2,n_3)).$$

Similarly, we can prove

$$CA(1,0)_x^{k_1} \diamond (CA(k_2,n_2) \cup CA(k_2,n_3)) = (CA(1,0)_x^{k_1} \diamond CA(k_2,n_2)) \cup (CA(1,0)_x^{k_1} \diamond CA(k_2,n_3)).$$

□

Example 19 *We note* $CA(3,3) = (\mathbf{Z}, \{0,1,2\}, \{\phi, \{0\}\})$ *and* $CA(3,102) = (\mathbf{Z}, \{0,1,2\}, \{\{0\}, \{1\}, \{0,2\}, \{1,2\}\})$. *The composition* $CA(3,3) \diamond CA(3,102) = (\{0,1,2,3,4\}, \{\{1\},\{0,1\},\{1,4\},\{0,1,4\}, \{3\},\{0,3\},\{3,4\},\{0,3,4\}\})$. *Since* $\{\{1\},\{0,1\},\{1,4\},\{0,1,4\},\{3\},\{0,3\},\{3,4\},\{0,3,4\}\}) = \bigcup\{\{\{0\} \cup s, s \cup \{4\}, \{0\} \cup s \cup \{4\}\} \mid s \in \{\{1\},\{3\}\}\}$, *we have* $CA(3,3) \diamond CA(3,102) = CA(3,18)_1^5$. *(cf. Fig 3, Fig 4, Fig 5)*

Fig. 3: An example of configurations of $CA(3,3)$

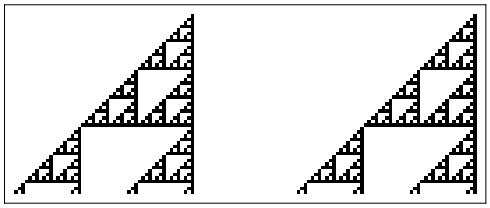


Fig. 4: An example of configurations of $CA(3,102)$

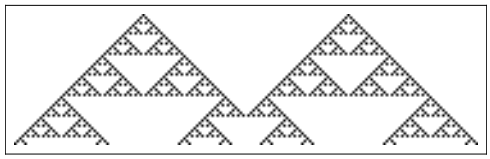


Fig. 5: An example of configurations of $CA(3,18) = CA(3,3) \diamond CA(3,102)$

Example 20 *A* 2*-neighborhood cellular automaton is considered as* 3*-neighborhood cellular automaton and* 3*-neighborhood cellular automaton is also considered as* 5*-neighborhood cellular automaton. The followings is an observation of the embeddings and compositions.*

$$\begin{aligned}
CA(2,1) &= (\mathbf{Z}, \{0,1\}, \{\phi\}) \\
CA(2,1)_0^3 &= (\mathbf{Z}, \{0,1,2\}, \{\phi, \{2\}\}) \\
&= CA(3,17) \\
CA(2,1) \diamond CA(2,1) &= (\mathbf{Z}, \{0,1,2\}, \{\{0,1\}, \{0,2\}, \{1,2\}, \{1\}\}) \\
&= CA(3,236) \\
CA(3,17) \diamond CA(3,17) &= (\mathbf{Z}, \{0,1,2,3,4,5\}, V') \\
&= CA(5, 3974950124) = CA(3,236)_0^5 \\
V' &= \bigcup\{\{s, s \cup \{3\}, s \cup \{4\}, s \cup \{3,4\}\} | s \in CA(3,236)\}
\end{aligned}$$

3 Generalization

A subset V of G is considered as a characteristic function $V : G \to 2$ where $2 = \{0, 1\}$. That is V is a function which values are

$$V(g) = \begin{cases} 0 & (g \notin V) \\ 1 & (g \in V). \end{cases}$$

Sometimes V is represented as an injection $i_V : V \to G$ where $i_V(g) = g$.

Extending our 2-states cellular automata on groups to many-states cellular automata on groups, we replace the set $2 = \{0, 1\}$ to a finite set S.

Definition 21 *Let G be a group, S a finite set. A generalized cellular automaton on G is a four-tuple $C = (G, S, i_V, V')$ of the group G, an injection $i_v : V \to G$, and a function $V' : S^V \to S$ where S^V is the set of all functions from V to S. A configuration $\mathbf{c} : G \to S$ is a function. The global transition function $F_C : S^G \to S^G$ is defined by $F_C(\mathbf{c})(g) = V'(\mathbf{c} \circ g \circ i_V)$.*

Proposition 22 *Let G be a group, and $S = 2 = \{0, 1\}$. The global function $F_C : 2^G \to 2^G$ is the same as defined in Definition 1. That is $F_C(\mathbf{c}) = \{g \in G \,|\, V'(\mathbf{c} \circ g \circ i_V) = 1\} = \bigcup_{g \in G} g \cdot l_{V'}(g^{-1} \cdot \mathbf{c} \cap V)$.*

Proof. For $g \in G$, we have

$$\begin{aligned} g \in \bigcup_{g \in G} g \cdot l_{V'}(g^{-1} \cdot \mathbf{c} \cap V) \quad &\Leftrightarrow \quad l_{V'}(g^{-1} \cdot \mathbf{c} \cap V) = \{e\} \\ &\Leftrightarrow \quad g^{-1} \cdot \mathbf{c} \cap V \in V' \\ &\Leftrightarrow \quad g^{-1}\{x \,|\, \mathbf{c}(x) = 1\} \cap V \in V' \\ &\Leftrightarrow \quad \{g^{-1}x \,|\, \mathbf{c}(x) = 1\} \cap V \in V' \\ &\Leftrightarrow \quad \{v \,|\, \mathbf{c}(gv) = 1\} \cap V \in V' \text{ (cf. } (x = gv)) \\ &\Leftrightarrow \quad \{v \,|\, \mathbf{c}(gv) = 1, v \in V\} \in V' \\ &\Leftrightarrow \quad \{v \,|\, \mathbf{c} \circ g \circ i_V(v) = 1\} \in V' \\ &\Leftrightarrow \quad V'(\mathbf{c} \circ g \circ i_V) = 1 \\ &\Leftrightarrow \quad g \in F_C(\mathbf{c}). \end{aligned}$$

□

Example 23 *Let $G = \mathbf{Z}$, $S = \mathbf{Z}_m$, and $V = \{-r, -r+1, \cdots, 0, \cdots, +r\}$. For a polynomial $f(X) = \sum_{i=-r}^{+r} a_i X^i$, ($a_i \in \mathbf{Z}_m$), we define the function $V'_{f(X)} : \mathbf{Z}_m^V \to \mathbf{Z}_m$ by $V'(x_{-r}, x_{-r+1}, \cdots, x_0, \cdots, x_{+r})$ $= \sum_{i=-r}^{+r} a_{-i} x_i$, ($(x_{-r}, x_{-r+1}, \cdots, x_0, \cdots, x_{+r}) \in {\mathbf{Z}_\mathbf{m}}^V$). A configuration $\mathbf{c} \in \mathbf{Z}_m^\mathbf{Z}$ is represented as a formal power series $\sum c_i X^i$ where $c_i = \mathbf{c}(i)$ (cf. Sato (1994); Manzini (1998)). Since $\mathbf{c} \circ j \circ i_V(i) =$*

$\mathbf{c}(j+i) = c_{j+i}$, *and* $\mathbf{c} \circ j \circ i_V = (c_{j-r}, c_{j-r+1}, \cdots, c_j, \cdots, c_{j+r})$, *we have*

$$\begin{aligned}
(\sum \mathbf{c}(i)X^i)f(X) &= (\sum c_i X^i)f(X) \\
&= (\sum c_i X^i)(\sum_{i'=-r}^{+r} a_{i'} X^{i'}) \\
&= (\sum c_i X^i)(\sum_{i'=-r}^{+r} a_{-i'} X^{-i'}) \\
&= \sum(\sum_{i'=-r}^{+r} c_i a_{-i'} X^{i-i'}) \\
&= \sum((\sum_{i'=-r}^{+r} a_{-i'} c_{j+i'})X^j) \text{ (cf. } j = i - i' \text{)} \\
&= \sum(V'(c_{j-r}, c_{j-r+1}, \cdots, c_j, \cdots, c_{j+r})X^j) \\
&= \sum(V'(\mathbf{c} \circ j \circ i_V)X^j). \\
&= \sum(F_C(\mathbf{c})(j)X^j).
\end{aligned}$$

The transition of the cellular automaton $C = (\mathbf{Z}, \mathbf{Z}_m, i_V, V'_{f(X)})$ *is corresponding to the product of polynomials (the formal power series).*

Acknowledgements

The authors thank Professor Yasuo Kawahara for his valuable suggestions and discussions. This work has been partially supported by Kyushu University Global COE Program "Education-and-Research Hub for Mathematics-for-Industry" and Regional Innovation Cluster Program (Global Type 2nd Stage) "Fukuoka Cluster for Advanced System LSI Technology Development".

References

E. Berlekamp, J. Conway, and R. Guy. *Winning Ways for Your Mathematical Plays, 2*. Academic Press, 1982.

M. Fujio. XOR$^2 = 90$ - graded algebra structure of the boolean algebra of local transistion rules -. In *RIMS kôkyûroku*, volume 1599, pages 97–102, 2008.

T. Ito. Abstract collision systems on g-sets. *J. of Math for Industry*, 2(A):57–73, 2010.

T. Ito, S. Inokuchi, and Y. Mizoguchi. An abstract collision system. In *Automata-2008 Theory and Applications of Cellular Automata*, pages 339–355. Luniver Press, 2008.

G. Manzini. Invertible linear cellular automata over $\mathbf{Z}_m$. *J. Comput. Syst. Sci.*, 56:60–67, 1998.

W. Pries, A. Thanailakis, and H. Card. Group properties of cellular automata and VLSI applications. *IEEE Trans. on Compters.*, C-35(12):1013–1024, 1986.

E. Rémila. An introduction to automata on graphs. In *Cellular Automata*, pages 345–352. Kluwer Academic Publishers, 1998.

Z. Róka. *Automates cellularies sur les graphes de Caley*. PhD thesis, Université Lyon l et Ecole Normale Supérieure de Lyon, 1994.

T. Sato. Group structured linear cellular automata over $\mathbf{Z}_m$. *J. Comput. Syst. Sci.*, 49:18–23, 1994.

J. von Neumann. *Theory of self-reproducing automata*. Univ. of Illinois Press, 1983.

T. Watanabe and S. Noguchi. On the uniform decomposition of automata and spatial networks. *IEICE Trans. Inf. and Syst.*, 11(2):1–9, 1982.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 265–276

Characterization of Single Hybridization in "Non-Interesting" class of Cellular Automata For SMACA Synthesis

Shiladitya Munshi[1†] and Sukanta Das[2‡] and Biplab K. Sikdar[3§]

[1]*Department of Computer Science and IT*
Meghnad Saha Institute of Technology, Kolkata, West Bengal, India 700150
[2]*Department of Information Technology*
Bengal Engineering and Science University, Shibpur, Howrah, West Bengal, India 711103
[3]*Department of Computer Science and Technology*
Bengal Engineering and Science University, Shibpur, Howrah, West Bengal, India 711103

This work investigates single hybridization in "Non-Interesting" class of Cellular Automata. Detailed analysis has been reported to model different criteria of single hybridization in this class of CA. The results establish that the "Non-Interesting" class of CA rules are the potential candidates in synthesizing Multiple Attractor Cellular Automata ($MACA$) with single length cycle attractor.

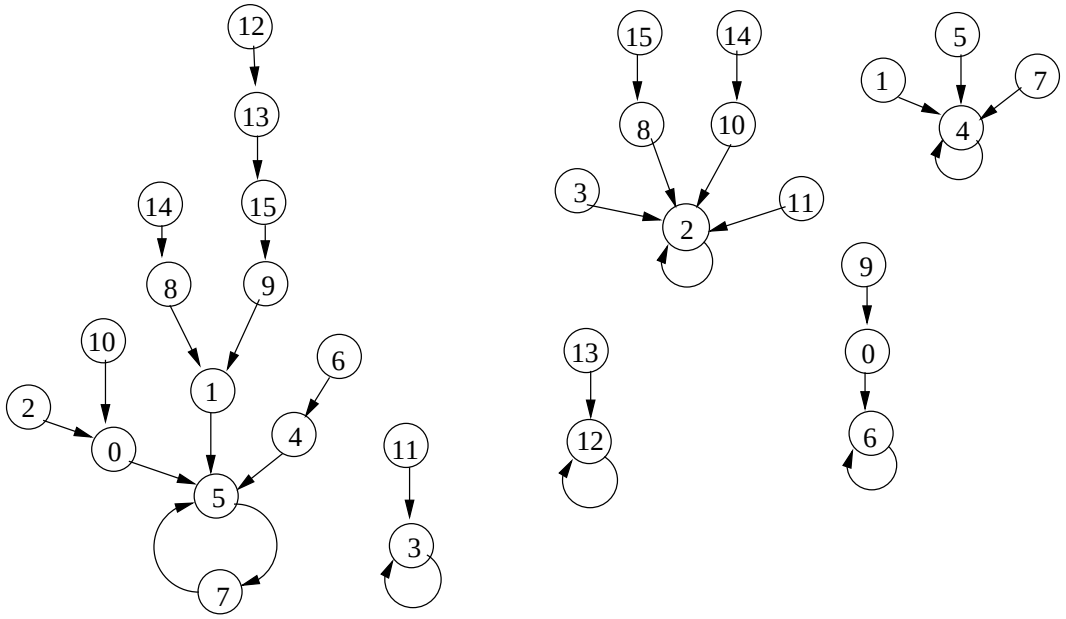
Keywords: Multiple Attractor Cellularr Automata **(MACA)**, Self Loop Multiple Attractor Cellularr Automata **(SMACA)**, Single Attractor Cellular Automata **(SACA)**

1 Introduction

Cellular Automata (CA) is a decentralized dynamical computing paradigm that evolves in discrete time and space. CA is represented as a spatially extended system, consisting of large numbers of simple identical components with local connectivity. The simple combinational logics employed at the local sites of CA give rise to a complex evolution of global states. The potential of CA to perform complex computations and its robustness has attracted a large section of researchers from diverse fields. Among many interesting and surprising global state transition patterns of CA evolution, researchers paid an immense interest towards a special class of CA referred to as the "Multiple Attractor Cellular Automata" ($MACA$).

The community of CA researchers has already acknowledged [4, 5, 8, 9, 10] the importance of $MACA$ structures in the field of pattern classification, design of associative memory, query processing, etc.

†shiladitya.munshi@yahoo.com
‡sukanta@it.becs.ac.in
§biplab@cs.becs.ac.in



(a) Phase space diagram for a CA with Rule Vector <40, 77, 40, 77>

(b) Phase space diagram for a CA with Rule Vector <40, 77, 77, 40>

Fig. 1: Sample $MACA$ (not all attractor being point state) and $SMACA$ (all attractors being point state)

The $MACA$ with single length (i.e point state) cycle called Self Loop $MACA$ ($SMACA$) has gained special attention in the recent years. The simple global phase space structure of $SMACA$ promises to be a better candidate for pattern classification, pattern recognition and other related applications due to reduced complexities in class identification.

The importance of $SMACA$ as pattern classifier [4], associative memory [5, 4] and other application areas [8, 9], as well as the absence of simple synthesis scheme for such a CA have motivated us to concentrate on the characterization of $SMACA$.

The characterization acts as the first step towards synthesis. It is done from the perspective of hybridization in "non interesting" class of elementary Cellular Automata rule space. The results of such characterization can be utilized to frame an efficient synthesis scheme for $SMACA$

Section II of this paper critically discusses the history and general notion about "Non Interesting" class of elementary Cellular Automata rule space followed by Section III that introduces the basic notations and terminologies related to $SMACA$. Finally, Section IV characterizes the effect of single hybridization on "Non Interesting" class of CA leading to the synthesis scheme for $SMACA$.

Tab. 1: Five Classes Elementary Cellular Automata Rules

$Class$	$RuleNumber$
Null	0, 8, 32, 40, 128, 136, 160, 168
Fixed Point	2, 4, 10, 12, 13, 24, 34, 36, 42, 44, 46, 56 57, 58, 72, 76, 77, 78, 104, 130, 132, 138, 140 152, 162, 164, 170, 172, 184, 200, 204, 232
Periodic	1, 3, 5, 6, 7, 9, 11, 14, 15, 19, 23, 25, 27, 28 29, 33, 35, 37, 38, 41, 43, 50, 51, 74, 108, 131 133, 134, 142, 156, 178
Locally Chaotic	26, 73, 154
Chaotic	18, 22, 30, 45, 54, 60, 90, 105, 106, 129, 137, 146, 150, 151

2 "Non Interesting" Elementary CA Rule Space

In his most influential paper [1], Wolfram reported the existence of four classes of rules for 2 state 3 neighborhood Cellular Automata (CA). This classification reported a class called Class I or Homogeneous Class where CA evolution led to a homogeneous state. Class I cellular automata evolve after a finite number of time steps to a unique homogeneous state.

This classification, enabled [2, 3], characterization of CA rule dynamics from the typical initial configurations. However, this classification was neither according to the dynamics from all initial configurations, nor it is the mathematical characterization. It could be thought of simple phenotype classification.

General notion about the class I rules admits the fact that the global state evolution dies out after a short span of time, thereby calling it "Non Interesting" from the computational complexity as well as universality perspective. But, critical and exhaustive computational experiments have proved that the "Non Interesting" CA rules can show some interesting properties if its phase transition length is considerably high and if the CA is initialized with a proper global configuration. Hence the "Non Interesting" is purely a qualitative measure and it depends of initial configuration1q of the CA.

Li et al have shown [3] that the 3 neighborhood CA rule space (containing 256 rules) can be folded down to Elementary rule space (containing 88 rules). Each rule in this Elementary space forms a cluster of 1, 2 or 4 rules, all of which share similar properties. In the present case, instead of taking the entire 3 neighborhood rule space,the Elementary rule space is considered for simplicity. For example, rule 40 is an elementary rule and it represents the rule cluster (40, 96, 235, and 249). Hence any discussion related with rule 40 equally implies to the entire cluster of (40, 96, 235, and 249). The work of Li, Packard and Langton [2, 3] has extended the Wolfram classification and pointed out five different classes. Among these, Class A ($Null$ rules) and Class B ($FixedPoint$ rules) are of great importance for the present work. Class A ($Null$ rules) and Class B ($FixedPoint$ rules) can be summarized as follow:

- Class A ($Null$ rules): CA evolution leads homogeneous fixed-point configurations.
- Class B ($FixedPoint$ rules): CA evolution leads to inhomogeneous fixed-point configurations.

Table 1 shows Li-Packard classification of CA rules with reference to the Elementary rule space.

Wolfram Type 1 Non Interesting class gets directly mapped with $Null$ and $FixedPoint$ classes of Li Packard classification. Both $Null$ and $FixedPoint$ class rules lead the CA evolution towards a fixed

attractor but the nature of attractors are different in both the cases. In $Null$ rules, the attractors are homogeneous that is point states or self loops, whereas in $FixedPoint$, the attractors may be composed of multi length cycles of states. Hence, for $SMACA$, $Null$ rules are the natural choice.

The pattern classification or pattern recognition solutions demand the patterns to be distributed over different dissipative phase space that could be identified by corresponding attractors. Here, the issues of universality or computational complexities generally do not play significant role. So the $Null$ rules, though do not show considerable computational complexities, can be thought for ideal candidate for $SMACA$ synthesis targeting different applications like pattern classification/recognition, associative memory implementations etc.

In the current paper, unless stated otherwise, the "Non Interesting" (NI) CA rules will always refer to the $Null$ class of Li Packard classification. The NI or $Null$ class CA rules can be characterized by the fact that within a phase space, all the states eventually move towards a single point state attractor or self loop. Hence, a $Null$ class CA can contain a cycle of length 1 at the max. Hence the dynamics of $Null$ class CA rules supports the basic criteria of $SMACA$. But at the same time, it lacks the presence of multiple dissipative phase space or basins as it contains only one basin. The presence of only one basin in the phase space confirms the CA to be referred to as Single Attractor Cellular Automata ($SACA$). The present work analyses the conditions for which a single hybridization in $Null$ class uniform CA transforms $SACA$ dynamics to $SMACA$ dynamics.

3 CA Preliminaries

A Cellular Automaton (CA) can be viewed as an autonomous finite state machine (FSM) consisting of a number of cells [6]. In a 3-neighborhood dependency, the next state $q^i_{(t+1)}$ of a cell is assumed to be dependent only on itself and on its two neighbours (left and right), and is denoted as
$q^i_{(t+1)} = f\ (q^{i-1}_{(t)}, q^i_{(t)}, q^{i+1}_{(t)})$ where $q^i_{(t)}$ represents the state of the i^{th} cell at t^{th} instant of time. f is the next state function and referred to as the rule of the automata. The decimal equivalent of the next state function, as introduced by Wolfram [7], is the rule number of the CA cell. For example

Rule 90: $q^i_{(t+1)} = q^{i-1}_{(t)} \oplus q^{i+1}_{(t)}$ where $\oplus$ function denotes modulo-2 addition. Since f is a function of 3 variables, there are 2^{2^3} i.e., 256 possible next state functions (rules) for a CA cell.

The CA is said to be uniform if the same rule have been introduced to each of the cells of a CA, otherwise it is termed as hybrid. In this paper we characterize the effect when hybridization occurs at i^{th} cell of a CA which is governed by the uniform $Null$ class rules. The following definitions are introduced to follow the reported characterization of null boundary CA.

Definition 3.1 Rule Vector (RV)*:The ordered sequence of Rules $< R_0, R_1 \cdots R_i \cdots R_{n-1} >$, of an n cell CA is referred to as its rule vector (RV) where i^{th} cell of CA is configured with rule R_i. If $R_0 = R_1 = R_i = R_{n-1}$, it is uniform RV other wise it is hybrid.*

Definition 3.2 Hybridization*: It is the process of introducing a non-homogeneous rule R_i at the i^{th} cell (i,e ith index of RV) of a uniform CA with RV $< R, R, R \cdots R >$. Single hybridization means introduction of one non homogeneous rule at i^{th} cell of the CA.*

Definition 3.3 ***Local Next State Function (f)****: The rule employed on a CA cell represents the local map that is, the local next state function f. Thus f_i refers to the local next state function corresponding to the rule R_i employed on i^{th} cell.*

Definition 3.4 Present and next state of a CA cell: *The present and next state of i^{th} cell is denoted as the a_i and b_i respectively. Hence $b_i = f_i(a_{i-1}, a_i, a_{i+1})$.*

Definition 3.5 Global next state function (F): *The global next state function F is derived from the local next state function as $F = (f_0 f_1 \cdots f_i \cdots f_{n-1})$ with the 3 variable Boolean function $f_i(a_{i-1}, a_i, a_{i+1})$.*

Definition 3.6 Global present and next state (A and B) : *The global present and next states are respectively denoted by A and B. Thus $A = (a_0, a_1 \cdots a_i \cdots a_{n-1})$, $B = F(A) = (b_0, b_1 \cdots b_i \cdots b_{n-1})$. That is B is the successor state of A, and A is the predecessor state of B.*

Definition 3.7 Self Loop Attractor (SLA) : *A state A is a SLA if $F(A) = B = A$. That is, there exists a cycle of length 1 with the state A.*

Definition 3.8 Rule Min Term (RMT): *The 8 Minterms of the 3 variable boolean function f_i corresponding to the rule R_i employed on i^{th} CA cell is referred to as RMTs. The three variables are a_{i-1}, a_i, a_{i+1}, the current states of $(i-1)^{th}$, i^{th}, $(i+1)^{th}$ cells respectively, The minterm $m = < a_{i-1}a_ia_{i+1} >$ is the RMT. The symbol T represents all the RMTs, whereby $T = T(0), T(1), T(2), T(3), T(4), T(5), T(6), T(7) = T(m)$. In general a single RMT for i^{th} cell is also denoted as $T^i \epsilon T$ where $T^i = < a_{i-1}a_ia_{i+1} >$.*

Definition 3.9 Compatible RMT Pair: *A pair of RMTs T^i and T^{i+1} in an RMT string $< \cdots T^{i-1}T^i T^{i+1} \cdots >$ (where $T^i \epsilon T$, $T^i = < a_{i-1}a_ia_{i+1} >$, and $T^{i+1} = < \acute{a_i}a\acute{}_{i+1}a\acute{}_{i+2} >$) are compatible if (i) $a_i = \acute{a_i}$ and (ii) $a_{i+1} = a\acute{}_{i+1}$.*

Example: Let us consider two RMTs T(2) and T(4) where T(2) = $< a_{i-1}a_ia_{i+1} > = < 010 >$ and T(4) = $< \acute{a_i}a\acute{}_{i+1}a\acute{}_{i+2} > = < 100 >$. Here $a_i = \acute{a_i} = 1$ and $a_{i+1} = a\acute{}_{i+1} = 0$. Hence the RMT pair T(2) and T(4) is compatible.

Let us consider another case with RMT pair T(3) and T(5) where T(3) = $< a_{i-1}a_ia_{i+1} > = < 011 >$ and T(5) = $< \acute{a_i}a\acute{}_{i+1}a\acute{}_{i+2} > = < 101 >$. Here $a_i = \acute{a_i} = 1$ but $a_{i+1} \neq a\acute{}_{i+1}$. Hence the RMT pair T(3) and T(5) is not compatible.

Definition 3.10 Valid RMT String: *A RMT string $< T^0 \cdots T^{i-1}T^iT^{i+1} \cdots T^{n-1} >$, representing the state of a CA is a valid RMT string if each pair T^i and T^{i+1} ($i = 0$ to $n-2$) is a compatible RMT pair.*

4 Characterization of Single Hybridization in "Non Interesting" CA

A "Non interesting" CA rule always produces only one cycle of length one, that is only one self loop. The valid RMT string that corresponds to this self loop is referred to as "Attractor RMT Sequence". For example, let us consider rule 40 ($Null$ rule , Table1). It has only one valid RMT string $< T(0)T(0)T(0)T(0) >$ which generates the attractor, hence, "Attractor RMT Sequence" for rule 40 is $< T(0)T(0)T(0)T(0) >$ or simply $< 0000 >$. The condition for a valid RMT string to be referred to as "Attractor RMT Sequence" is given as follows

Property 1: For an n length uniform CA with $Null$ rule R, the valid RMT String $S = < T^0 \cdots T^{i-1}T^i T^{i+1} \cdots T^{n-1} >$ is said to be "Attractor RMT Sequence" if and only if, for all i^{th} cell (i = 0 to n-1), the condition $a_i = b_i$ is true (where T^ii ϵ T , $T^i = < a_{i-1}a_ia_{i+1} >$, and b_i is the next state of i^{th} cell, hence $b_i \epsilon$ 0,1).

The RMTs that constitute "Attractor RMT Sequence" for rule R, form a set $ATTR\_SEQ_R$. Following algorithm identifies the "Attractor RMT sequence", given a $Null$ rule for n length CA.

Algorithm 1: $Identify\_ARS$
Input: A $Null$ rule R for an n length CA and an index j set to 0
Output: An n length "Attractor RMT sequence" $< T^0 \cdots T^{j-1} T^j T^{j+1} \cdots T^{n-1} >$
Step 0: for R, pick up the RMTs which holds Property 1
Step 1: for each picked up RMT T in Step 0, do Step 2
Step 2: while j is less than n do the following
Step 3: append T^j to the "Attractor RMT Sequence" $< T^0 \cdots T^{j-1} >$
Step 4: compute two compatible RMTs T^{j+1} and $T^{j'+1}$ from T^j
Step 5: for each of RMTs T^{j+1} and $T^{j'+1}$, computed at Step 4, if it holds Property 1, do Step 2 with j ← j+1, else do Step 6
Step 6: start another execution as per Step 1
Step 7: output "Attractor RMT sequence" $< T^0 \cdots T^{j-1} T^j T^{j+1} \cdots T^{n-1} >$ and Stop.

Step 2 to Step 5 of $Identify\_ARS$ algorithm executes n times and this execution is done for all of the RMTs satisfying Property 1. For a specific $Null$ rule R, the number of RMTs satisfying Property 1 is constant (say k) and hence the entire algorothm runs with a time complexity of $k.n$

In a state space, a cycle of length k can be thought of as a point state or self loop attractor at k^{th} temporal extension. In case of $Null$ class CA rules, the RMTs with relation a_i = b_i are so arranged and distributed that there exists no k^{th} temporal extension at which, for an n cell Cellular Automata, for all i^{th} cell (i = 0 to n-1), the condition a_i = b_i is true. Hence $Null$ class rules are devoid of any attractor of length k (k = 2 to n-1).

This property, though not the outcome of this study, guides the hybridization in "Non interesting" CA rules. The possibility of formation of multi-length cycle due to the introduction of single hybridizing rule gets nullified in the presence of NULL class CA rules.

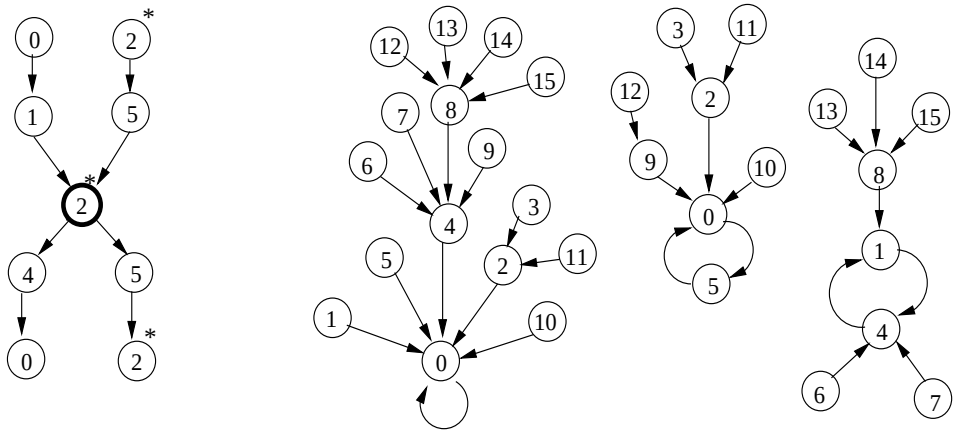
4.1 Characterization

To proceed with the characterization, let us define the following key concepts

Definition 4.1 Non Point RMT Set*: Non Point RMT Set for rule R N_PTR is defined as the set of RMTs of rule R for which the condition a_i = b_i is not true.*

Definition 4.2 RMT Transition Tree*: For an n cell CA of rule R, it is a non-linear Binary Tree with n levels (0 to n-1), in which a node containing one RMT is connected with another node containing compatible RMT. As one RMT always generates a compatible RMT pair as per Definition 3.9, a parent node will always contain exactly two child nodes excepting at level n-1. At level n-1, no odd RMTs will be present due to null boundary condition. The RMT transition tree is always drawn with reference to a specific RMT at jth cell of CA (j = 0 to n-1). Hence the sub tree, all the complete path of which passes through specific RMT at level j corresponds to the reference RMT. The nodes of the tree also hold the status of a_i = b_i relation.*

Let us consider Rule 8(00001000). The next state of all the RMTs excepting T(3) is 0 where as next state of T(3) is 1. Hence T(2), T(6) and T(7) do not hold a_i = b_i relation. Hence N_PT8 is T(2), T(6), T(7) or simply 2,6,7.



(a) RMT Transition Tree for 5 cell CA with rule 8 and with reference to T(2) at 2nd level

(b) Phase space diagram of RV <8, 24, 8, 8 >

(c) Phase space diagram of RV <8, 9, 8, 8>

Fig. 2: Illustration of Example 1 and 2

Figure 2(a) represents a RMT transition tree (for 5 cell CA with rule 8) with reference to RMT T(3) at 3^{rd} cell or at j = 2 where j = 0 to 4. The reference node marked with bold line and * denotes the node that does not follow $a_i = b_i$ relation.

The following theorems lead to characterization of the hybridization in "Non interesting" CA rules.

Theorem 4.1 *For a pair of hybridizing rule H and uniform rule R, if the relation $N\_PT_R \subset N\_PT_H$ is true, then the hybridization will never lead to SMACA formation.*

Proof: The singletone existence of Self Loop Attractor for rule R is due to violation of the relation $a_i = b_i$ for RMT T where $T \epsilon N\_PT_R$. As $N\_PT_R \subset N\_PT_H$ holds good, the same violation of the relation $a_i = b_i$ exists in the hybridized CA. Hence the singletone existence of Self Loop Attractor never changes and the hybridization preserves SACA nature not leading to SMACA. Hence proved. □

Example : Let us consider, R = 8 (00001000) and H = 24 (00011000). Hence N_PT_R = T(2), T(6), T(7) or simply N_PT_R = 2,6,7 and N_PT_H = T(2),T(4),T(6),T(7) or simply 2,4,6,7. Hence $N\_PT_R \subset N\_PT_H$ is true. This hybridization will never lead to SMACA as the same can be reviewed and confirmed from Figure 2(b).

Theorem 4.2 *For a pair of hybridizing rule H (at i^{th} cell) and uniform rule R, the hybridization will never lead to $SMACA$ formation, if the following conditions are true:*
a) $N\_PT_H = N\_PT_R \cup S_H$ where $S_H \cap ATTR\_SEQ_R \neq Phi$; and
b) At least one element of S_H must be present at i^{th} position of "Attractor RMT Sequence".

Proof: The condition $N\_PT_H = N\_PT_R \cup S_H$ dictates that there is no possibility of formation of auxiliary "Attractor RMT Sequence" in the hybridized CA. It is due to the fact that RMTs non conforming to

the relation $a_i = b_i$ in R have not been changed in H. In addition to this, the existing "Attractor RMT Sequence" corresponding to R gets disturbed due to the fact that RMT(s) say T, from $ATTR\_SEQ_R$ does not hold the relation $a_i = b_i$ any more in H as $S_H \cap ATTR\_SEQ_R \neq \Phi$ and $S_H \subset N\_PT_H$. So at the hybridizing cell site, "Attractor RMT Sequence" will fail to hold $a_i = b_i$i relation resulting no formation of Self Loop Attractor . So as (i) "Attractor RMT Sequence" of uniform CA with R exists no more and (ii) No possibility of Self Loop formation evolves up due to introduction of rule H, no self loop can be generated for the hybridized CA and hence $SMACA$ generation is impossible. Proved. □

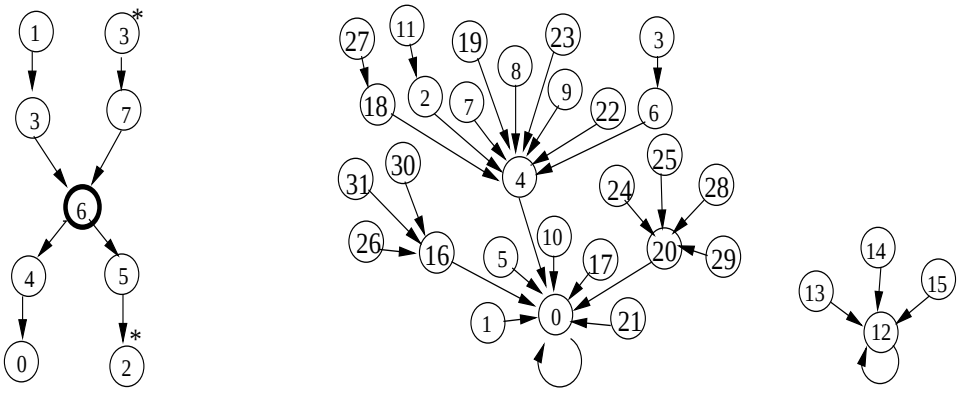
Example : Let us consider, R = 8 (00001000) and H = 9 (00001001). For this pair of rules, $ATTR\_SEQ_R$ = T(0) = 0, N_PT_R = T(2), T(6), T(7) = 2,6,7 and N_PT_H = T(0), T(2), T(6), T(7) = 0, 2,6,7. Mathematically, $N\_PT_H = N\_PT_R \cup S_H$ where S_H = 0. So there exists an RMT T = T(0) = 0 for which T ϵ S_H and T ϵ $ATTR\_SEQ_R$ is true. Hence this hybridization will never lead to $SMACA$. This can be rechecked with the phase space diagram of Figure 2(c).

Above discussion analyses the conditions which refuse a hybridizing rule H to be introduced into a uniform CA with rule R in order to possible $SMACA$ generation. Hence the rule non conforming to these conditions stated in theorem 4.1 and theorem 4.2 has a potential to form $SMACA$ structure when introduced to a uniform CA with "Non Interesting CA" rule R. One of such rules can be referred to as "Potential Rule" H_P. A "Potential Rule" H_P can be characterized by the property $N\_PT_R \not\subset N\_PT_{H_P}$.

Theorem 4.3 *An n cell uniform CA with rule R, when hybridized with potential rule H_P at i^{th} level or cell, leads to $SMACA$ with k+1 number of Self Loop Attractors if and only if there exists k number of complete path (each node satisfying $a_i = b_i$ relation) from 0^{th} level to $(n-1)^{th}$ level of RMT Transition Tree of the hybridized RV with reference to RMT (say T) at level i; where T $\subset (N\_PT_R - N\_PT_{H_P})$.*

Proof: A Potential Rule H_P confirms that there lies at least Υ (Υ = 1, 2 $\cdots$) such RMTs that do satisfy the relation $a_i = b_i$ but that used not to satisfy the same in uniform CA rule R. So the "RMT Transition Tree" constructed with reference to those Υ no of RMTs of R at i^{th} level, could not produce any auxiliary "Attractor RMT Sequence". This is due to the fact that in those case, no complete path could be identified from 0^{th} level to $(n-1)^{th}$ level with the relation $a_i = b_i$ satisfied at each level. But in H_P, these Υ no of RMTs maintain the relation $a_i = b_i$. Hence for a fresh "RMT Transition Tree", with reference to Υ number of RMTs of H_P, at i^{th} cell/level. If there are k number of complete paths, then each of these paths, (satisfying $a_i = b_i$ all along) will always yield k no of Self Loops or Point States. The complete paths identified, itself confirms the identification of Self Loop or Point State attractor as $< b_0 b_1 \cdots b_i \cdots b_{(n-1)} >$ = $< a_0 a_1 \cdots a_i \cdots a_{(n-1)} >$ enabling synthesis of a $SMACA$ with k + 1 no of basins grown around k +1 no of Self Loops or Point States. The extra one attractor is accounted for the original Self Loop Attractor of Uniform CA with R. Hence Proved. □

Example : Let us consider R = 8(00001000) and H = 90(01011010). Here N_PT_R = T(2), T(6), T(7) = 2,6,7 and N_PT_H = T(1), T(2), T(4),T(7) = 1, 2, 4, 7. As $N\_PT_R \not\subset N\_PT_H$, rule 90 can be treated as H_P . Hence N_PT_R - $N\_PT_{H_P}$ yields 6. Let us further consider that a 5 cell uniform CA with rule 8 is hybridized with rule 90 at i = 2 or in 3^{rd} cell. Now Figure 3(a) shows the RMT Transition Graph of this 5 cell hybridized CA, with reference to RMT T(6) or simply 6 at level i = 2. The Figure 3(a) shows that only one complete path with condition $a_i = b_i$ satisfied all along can be identified as <T(1) T(3) T(6) T(4) T(0)> = <13640> (shown in bold arrows). This auxiliary Attractor RMT Sequence generates the Self Loop or Point State as $< a_0 a_1 a_2 a_3 a_4 >$ = <01100> = 12. Hence 5 cell hybridized $CA < 8, 8, 90, 8, 8 >$ will lead to two basin $SMACA$ with two point state attractors (one being 6 and the other being original



(a) RMT Transition Tree of hybridized CA <8, 8, 90, 8, 8> W.R.T RMT T(6) at 2nd level

(b) Phase Transition Diagram for <8, 8, 90, 8, 8>

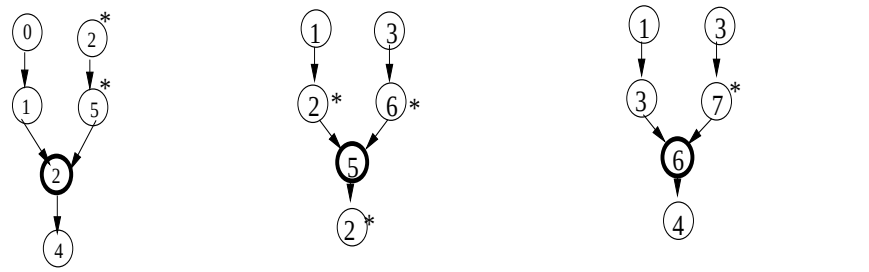
Fig. 3: Illustration of Example 3

attractor 0) . The phase transition diagram of CA with RV $< 8, 8, 90, 8, 8 >$ (as depicted in Figure 3(b)) confirms the result.

Theorem 4.4 *An n cell uniform CA with rule R, when hybridized with "Potential Rule" H_P at i^{th} level or cell, leads to destruction of original Self Loop or Point State attractor corresponding to R if $N\_PT_{H_P} \cap ATTR\_SEQ_R \neq \emptyset$ and there lies at least one element in $N\_PT_{H_P}$ that is present at i^{th} position of "Attractor RMT Sequence".*

Proof: The relation $N\_PT_{H_P} \cap ATTR\_SEQ_R \neq \emptyset$ implies that the hybridizing rule H_P has some RMT(s) say T which does not satisfy the relation $a_i = b_i$, and at the same time those RMT(s) T also exists in i^{th} position within "Attractor RMT Sequence" corresponding to R. Hence the "Attractor RMT Sequence" will not exist any more as the primary condition of being Self Loop State or Point State ($a_i = b_i$) gets hampered at i^{th} position. So the original Self Loop or Point State exists no more as a direct result of destruction of original "Attractor RMT Sequence". □

Example : Let us consider a 4 cell uniform CA with rule 40 (00101000) that has to be hybridized with rule 77 (01001101) at level i = 2. Hence R = 40 and H = 77. N_PT_R = 2,5,6,7 and N_PT_H = 0,7 are computed and as $N\_PT_R \not\subset N\_PT_H$ is true, H can be treated as H_P. Here $ATTR\_SEQ_R$ = 0 and $N_PT_{H_P} \cap ATTR\_SEQ_R \neq \emptyset$ =0, so the original Self Loop State or Point State 0 will not be preserved in Hybridized CA. Now N_PT_R - $N\_PT_{H_P}$ yields 2, 5, 6. Figure 4(a), (b) and (c) show the RMT Transition Tree with reference to RMTs 2, 5 and 6 respectively at i = 2. Figure 4(b) reveals that no auxiliary Attractor RMT Sequence generates due to RMT 5, whereas due to RMT 2 and RMT 6, two auxiliary Attractor RMT Sequence are generate, namely, $< T(0)T(1)T(2)T(4) >$ and $< T(1)T(3)T(6)T(4) >$



(a) RMT Transition Tree W.R.T T(2) (b) RMT Transition Tree W.R.T T(5) (c) RMT Transition Tree W.R.T T(6)

Fig. 4: RMT Transition Tree of Hybridized CA <40, 40, 77, 40>, With Reference To RMT T(2), T(5) and T(6) at level i= 2

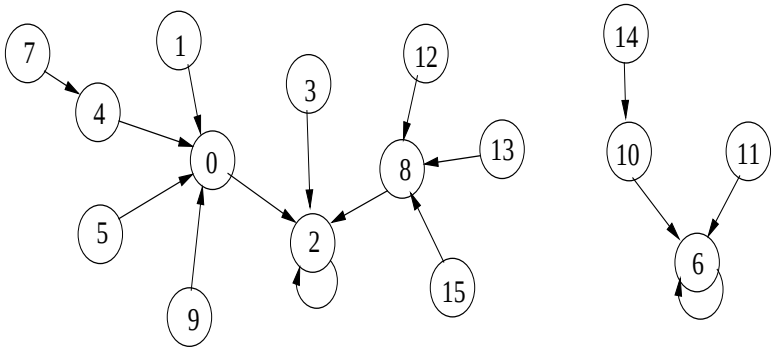


Fig. 5: Phase Transition Diagram for CA with RV <40, 40, 77, 40>

respectively. These two auxiliary Attractor RMT Sequence yields state 2 (0010) and 6 (0110) as two attractors. Hence as the original attractor has been destructed and as two new attractors have been generated, so as a result, the newly formed $SMACA$ structure of $CA < 40, 40, 77, 40 >$ will have two basins grown around two Self Loop or Point State 2 and 6. This can be verified from Figure 5 which depicts the phase transition diagram of $< 40, 40, 77, 40 >$.

5 Conclusion

The present work on characterization of single hybridization in "non-interesting" uniform CA, sets a platform for simple architecture of two class pattern classifier. The insights developed can be utilized further in order to have simple and robust algorithm for $SMACA$ synthesis. Characterization of basin volume or height or even cross basin transition of states may produce useful contribution in that regard.

References

[1] S. Wolfram, 1984. *Universality and complexity in cellular automata*,Physica D, -10, pp1-35.

[2] W. Li, N. Packard, and C. Langton, September 1990. *Transition phenomena in cellular automata rule space*, Physica D, 45(1-3), 77 - 94.

[3] W. Li and Norman Packard , 1990. *The Structure of the Elementary Cellular Automata Rule Space*, Complex Systems, Vol-4, pp 281-297.

[4] Pradipta Maji, Chandrama Shaw, Niloy Ganguly, Biplab K. Sikdar, and P. Pal Chaudhuri , December 2003 .*Theory and Application of Cellular Automata for Pattern Classification*, Fundamenta Informaticae, 58(3-4), pp. 321–354.

[5] Niloy Ganguly, Pradipta Maji, Biplab K. Sikdar, and P. Pal Chaudhuri , November 2002. *Generalized Multiple Attractor Cellular Automata (GMACA) Model for Associative Memory*,International Journal of Pattern Recognition and Artificial Intelligence, 16(7), pp. 781–795.

[6] Neumann, J. V , 1966. *The Theory of Self-Reproducing Automata*, A. W. Burks, Ed. University of Illinois Press,Urbana and London.

[7] Wolfram, S , 1986. *Theory and Application of Cellular Automata*, World Scientific, 1986.

[8] Sikdar, B. K., Ganguly, N., Majumder, P., Chaudhuri, P. P , January 2001. *Design of Multiple Attractor GF(2^p) Cellular Automata for Diagnosis of VLSI Circuits*, Proceedings of 14th International Conference on VLSI Design, India, January 2001, pp 454 - 459.

[9] Pal, K, December 1998. *Theory and Application of Multiple Attractor Cellular Automata for Fault Diagnosis*,Proceedings of Asian Test Symposium.

[10] Sikdar, B. K., Ganguly, N., Karmakar, A., Chowdhury, S. S., Pal Chaudhuri, P, November 2001. *Multiple Attractor Cellular Automata for Hierarchical Diagnosis of VLSI Circuits*, 5Proceedings of 10th Asian Test Symposium.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 277–286

A Generalization of Automorphism Classification of Cellular Automata

Hidenosuke Nishio

Kyoto University
Iwakura Miyake-cho 204-1, Sakyo-ku, 606-0022, Kyoto, Japan
email: yra05762@nifty.com

In this paper we make a generalization (called g-automorphism) of the automorphism of cellular automata (CA for short) introduced by H. Nishio in 2009. At defining g-automorphism, we consider both permutations of the position and the value of the arguments of the local function with relevant permutation of the neighborhood. We prove that the g-automorphisms constitutes a group under a rule of the semi-direct product. The group acts on the local function and naturally induces a classification of CA. Every CA in a class has the same global property *up to permutation*. For explaining the idea we preferably use rule 110 which has been proved computation universal and calculate all g-automorphisms of f_{110}. As a byproduct we assert that there are 48 universal functions up to permutation. Finally we show the g-automorphism classification of 256 local functions of ECA into 11 classes.

Keywords: cellular automaton, automorphism, classification, permutation, neighborhood, semi-direct product

1 Introduction

In the history of the cellular automaton (CA for short), most studies first assume some standard neighborhood (von Neumann, Moore) and then investigate the global behaviors and mathematical properties or look for a local function that would solve a given problem, say, the self-reproduction, the Game of Life and so on. One could, however, ask a question: What happens if the neighborhood is changed from the standard.

Around 1997 H. Nishio showed that infinitely many CA are made by changing the neighborhood with a fixed local function (4) and Th.Worsh and H.Nishio proposed a computation universal CA which achieves universality by changing the neighborhood (11).

Another direction of the research which will attack this question is to make clear the conditions for CA to be the *same* or equivalent when the neighborhood is changed. Suppose that CA is defined by a 4 tuple $(\mathbb{Z}^d, Q, f, \nu)$, where $\mathbb{Z}^d$ is the d-dimensional Euclidean cellular space, Q is the set of cell states, f is the local function and ν is the neighborhood, which is a mapping from $\{1, ..., n\}$ to $\mathbb{Z}^d$. The i-th neighbor $\nu(i),\ 1 \leq i \leq n$ is connected to the i-th argument of f. When the space $\mathbb{Z}^d$ and the state set Q are understood, the global behavior of CA is determined by its *local structure* (f, ν). Two local structures are

called *equivalent* if and only if they induce the same global functions. As for equivalence we particularly proved a basic theorem: *Two CA are equivalent if and only if their local structures are permutation of each other* (7). See Fig. 1.

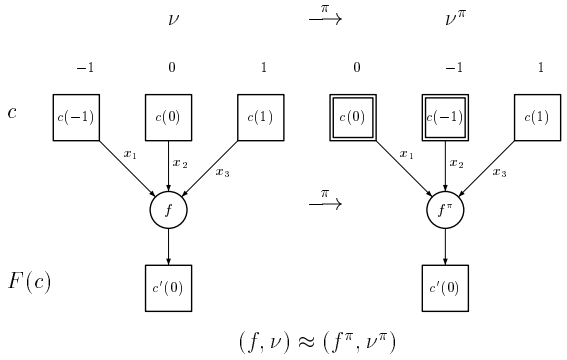


Fig. 1: Permutation equivalence of 1-dimensional 3-neighbor CA

Based on this theory of the permutation equivalence of local structures, we defined the automorphism for local structures and investigated the automorphism classification of the local functions (5; 6) . First we consider the equivalence and then we defined the automorphism as the equivalence plus a permutation φ of the states (value of arguments). To explain the point, take for example a computation universal ECA $(f_{110}, (-1, 0, 1))$, where $f_{110} = x_1x_2x_3 + x_2x_3 + x_2 + x_3$ in the polynomial expression, see Section 2.3. Then by a permutation of the arguments $\pi_2 = (12)$ (transpose x_1 with x_2), we obtain $f_{110}^{\pi_2} = x_1x_2x_3 + x_1x_3 + x_1 + x_3 = f_{122}$. The function f_{122} is probably not universal on $(-1, 0, 1)$, $(0, -1, 1)$ and others. But, by inversely permuting the local structure $(f_{122}, (0, -1, 1))$ with $\pi_2^{-1}(= \pi_2)$, we have $(f_{122}^{\pi_2}, (0, -1, 1)^{\pi_2}) = (f_{110}, (-1, 0, 1))$, that is $(f_{122}, (0, -1, 1))$ is equivalent with $(f_{110}, (-1, 0, 1))$. In this sense, rule 122 is called universal *up to permutation*.

Next we defined the automorphism: (f', ν') is called *automorphic* with (f, ν) if and only if there is a pair of permutations ν and φ such that $(f', \nu') = (\varphi^{-1} f^\pi \varphi, \nu^\pi)$. For example, if we permute $(f_{110}, (-1, 0, 1))$ with π^2 and $\varphi = (1, 2)$ (transposition of states 0 and 1) [1], we have $(f_{161}, (0, -1, 1))$. Therefore f_{161} is also universal *up to permutation*. In what follows we often omit the suffix *up to permutation*.

Now we generalize the automorphism of CA in such a way that every argument of f is permuted independently. The local function is expressed by a polynomial in n variables $f(\mathbf{x}_n) = f(x_1, ..., x_n)$ over finite field $GF(q)$ and the set of such polynomials is denoted $\mathcal{P}_{n,q},\ 1 \leq n, 2 \leq q$. We are going to define the g-automorphism for $\mathcal{P}_{n,q}$. For two CA A and A', $A' = (f', \nu')$ is called *g-automorphic* with $A = (f, \nu)$ denoted $A \cong_g A'$, if and only if there is a 3-tuple of permutations $(\pi, \psi, \varphi(n))$ such that $(f', \nu') = (f^\pi, \psi f^\pi \varphi(n))$, where $\varphi(n) = (\varphi_1, ..., \varphi_n)$ and $\varphi_i, 1 \leq i \leq n$ permutes the value of the i-th

[1] $\varphi^{-1} f \varphi$ is called *conjugation*, see Subsection 2.3

argument, see Fig.2.

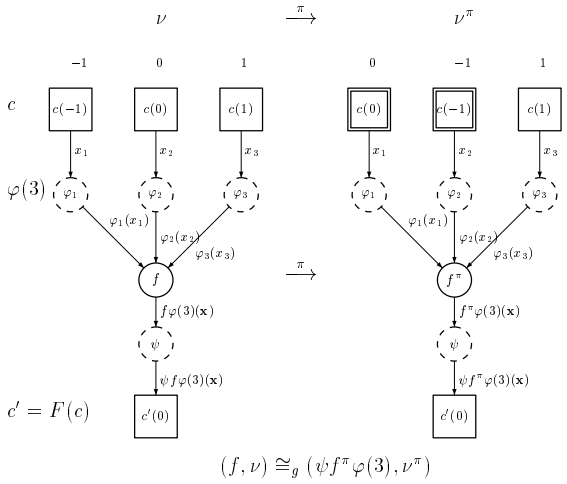


Fig. 2: g-automorphism of 1-dimensional 3-neighbor CA

The set of automorphisms $G_{n,q} = \{(\pi, \psi, \varphi(n))|\pi \in S_n, \psi \in S_q, \varphi(n) \in S_q^n\}$ is proved a group under the group operation of semi-direct product. The g-automorphism group acts on $\mathcal{P}_{n,q}$ and induces a classification of CA such that every CA in a class has the same global property *up to permutation*. For explaining the idea we preferably use rule 110 a computation universal ECA. To be specific we show that there are 48 functions which are universal up to permutation. This is compared with 6 ECA which are automorphic with f_{110} (5; 6).

Finally we show the g-automorphism classification of ELFs in the form of a table, where every g-automorphism class (GN class for short) is expressed by a union of several NW classes obtained by H. Nishio (5). It is seen that 256 ELF are classified into 11 GN classes, which is compared with 46 NW classes.

This work has been inspired by the past mathematical works about the logical circuits made by C. Shannon (9),D. Slepian (10) and M. Harrison (2) during 1950s the dawn of the computer science. Specifically they formulated and generally solved the problem of counting the number of the equivalent or symmetry classes of Boolean functions by use of the Pólya's counting theory. However, their motivation for such an investigation was fairly different from ours. They aimed at elucidating the physical/structural similarity of logical circuits from the point of view of the technological design . They argued that the *cost* of the circuit is invariant when permuting and/or complementing one or more variables. In our terminology the Boolean functions belonging to the same class are g-automorphic. Mathematically speaking, their theory is exclusively concerned with the Boolean functions ($\mathcal{P}_{n,2}$) and even afterward, as far as I know, has not been generalized to arbitrary functions ($\mathcal{P}_{n,q}$).

2 Preliminaries

The definitions and previous results are briefly restated, of which details will be found in (7; 5; 6).

2.1 CA and local structures

A cellular automaton is defined by a 4-tuple $(\mathbb{Z}^d, Q, f, \nu)$, where $\mathbb{Z}^d$ is a d-dimensional Euclidean space, Q is a finite set of *cell states*, $f : Q^n \to Q$ is a *local function* and ν is a *neighborhood*.

- **[neighborhood]** *A neighborhood* is a mapping $\nu : \mathbb{N}_n \to \mathbb{Z}^d$, where $\mathbb{N}_n = \{1, \ldots, n\}$ and $n \in \mathbb{N}$. This can equivalently be seen as a list ν with n components $(\nu_1, \ldots, \nu_n)$, where $\nu_i = \nu(i), 1 \leq i \leq n$, is called the i-th *neighbor*. The i-th argument of f is connected to the i-th neighbor.
- **[local structure]** A pair (f, ν) is called a *local structure* of CA. We call n the *arity* of the local structure. When the space $\mathbb{Z}^d$ and the state set Q are understood, CA is often identified with its local structure.
- **[global function]** A local structure uniquely induces a *global function* $F : Q^{\mathbb{Z}^d} \to Q^{\mathbb{Z}^d}$, which is defined by
$$F(c)(x) = f(c(x + \nu_1), ..., c(x + \nu_n)), \tag{1}$$
for any *global configuration* $c \in Q^{\mathbb{Z}^d}$, where $c(x)$ is the state of cell $x \in \mathbb{Z}^d$ in c.

Remark 1 *In the previous paper (7) the definition of local structures was more general, but in this paper we assume, without loss of generality, a restricted but most usual case of* reduced *local structures, see the following definition and Lemma 1.*

2.2 Previous results on the equivalence of local structures

Here we extract from the previous papers some basic results on the equivalence of local structures, which entail the present work on the generalized automorphism.

Definition 1 ***[reduced local structure]*** *A local structure is called* reduced*, if and only if*

- *ν is injective, i.e. $\nu_i \neq \nu_j$ for $i \neq j$ in the list of neighborhood ν and*
- f depends on all arguments.

Lemma 1 *For each local structure (f, ν) there is an equivalent local structure (f', ν') which is reduced.*

Definition 2 ***[equivalence]*** *Two local structures (f, ν) and (f', ν') are called* equivalent*, if and only if they induce* the same global function*. In that case we write $(f, \nu) \approx (f', \nu')$.*

Definition 3 ***[permutation of local structure]*** *For $\pi \in S_n$ we define the permutation of the local function and neighborhood by*
$$f^\pi(x_1, ..., x_n) = f(x_{\pi(1)}, ..., x_{\pi(n)}) \tag{2}$$
and
$$\nu^\pi = (\nu^\pi_1, ..., \nu^\pi_n), \text{where } \nu^\pi_{\pi(i)} = \nu_i,\ 1 \leq i \leq n. \tag{3}$$

Then we have the basic properties of the permutation of local structures.

Lemma 2 *(f, ν) and (f^π, ν^π) are equivalent for any permutation π.*

Theorem 1 ***[permutation-equivalence of local structures]***
If (f, ν) and (f', ν') are two reduced local structures which are equivalent, then there is a permutation π such that $(f^\pi, \nu^\pi) = (f', \nu')$.

2.3 Some technical notes

(1) The local function is expressed by a polynomial in n variables $f(\mathbf{x}_n) = f(x_1, ..., x_n)$ over finite field $GF(q)$ and the set of such polynomials will be denoted $\mathcal{P}_{n,q},\ n \geq 1, q \geq 2$. $\mathcal{P}_{n,q}$ is a polynomial ring over $GF(q) \mod (x_1^q - x_1) \cdots (x_n^q - x_n)$. Obviously $|\mathcal{P}_{n,q}| = q^{q^n}$. For small n and q, f is written as follows.

- If $f \in \mathcal{P}_{3,q}$,

$$f(x_1, x_2, x_3) = u_0 + u_1x_1 + u_2x_2 + \cdots + u_ix_1^hx_2^jx_3^k + \cdots + u_{q^3-2}x_1^{q-1}x_2^{q-1}x_3^{q-2} + u_{q^3-1}x_1^{q-1}x_2^{q-1}x_3^{q-1}, \quad \text{where } u_i \in GF(q),\ 0 \leq i \leq q^3 - 1. \quad (4)$$

- The local function of an ECA is called the *elementary local function* denoted ELF, which is generally expressed by a polynomial $f(x_1, x_2, x_3)$ over $GF(2)$ as shown below.

$$f(x_1, x_2, x_3) = u_0 + u_1x_1 + u_2x_2 + u_3x_3 + u_4x_1x_2 + u_5x_1x_3 + u_6x_2x_3 + u_7x_1x_2x_3, \quad \text{where } u_i \in GF(2) = \{0, 1\}, 0 \leq i \leq 7. \quad (5)$$

  Note that for $f \in \mathcal{P}_{3,2}$, the polynomial expression is equivalently transformed to the Boolean expression by $a + b + ab$ (polynomial) $= a \vee b$ (Boolean), ab (polynomial) $= a \wedge b$ (Boolean) and $a+1$ (polynomial) $= \overline{a}$ (Boolean). Conjugation $f' = \varphi_1^{-1} f \varphi_1 = f(x_1+1, x_2+1, x_3+1)+1$

- In the sequel, every ELF is numbered by a so called Wolfram number such as $f_{110} = x_1x_2x_3 + x_2x_3 + x_2 + x_3$. The Java program **catest106d** made by C.Lode (3) contains a useful tool for conversion between the Boolean, the polynomial and the Wolfram number.

(2) Permutations of 3 objects are usually expressed by a symmetric group $S_3 = \{\pi_i, 0 \leq i \leq 5\}$ as is shown below.

$$\pi_0 = \mathbf{1} = \begin{pmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{pmatrix},\ \pi_1 = (23) = \begin{pmatrix}1 & 2 & 3\\ 1 & 3 & 2\end{pmatrix},\ \pi_2 = (12) = \begin{pmatrix}1 & 2 & 3\\ 2 & 1 & 3\end{pmatrix},$$

$$\pi_3 = (123) = \begin{pmatrix}1 & 2 & 3\\ 2 & 3 & 1\end{pmatrix},\ \pi_4 = (132) = \begin{pmatrix}1 & 2 & 3\\ 3 & 1 & 2\end{pmatrix},\ \pi_5 = (13) = \begin{pmatrix}1 & 2 & 3\\ 3 & 2 & 1\end{pmatrix}$$

Note that S_3 is not commutative: $\pi_2\pi_1 = (12)(23) = (123) = \pi_3$ but $\pi_1\pi_2 = (23)(12) = (132) = \pi_4$. The neighborhood $(-1, 0, 1)$ of ECA is called the elementary neighborhood (ENB for short). Then $ENB^{\pi_1} = (-1, 1, 0)$, $ENB^{\pi_2} = (0, -1, 1)$ and so on.

3 (n, q)-permutation of local functions

We define two kinds of permutations called *n-permutation* and *q-permutation* of the local function and then unify them as *(n,q)-permutation* of f.

1. **Definition 4** ***[n-permutation of f]*** *The permutation of f defined in Definition 3 is essentially related to a permutation of the neighborhood and called hereafter the* n-permutation *of f.*

$$f^{\pi}(x_1, ..., x_n) = f(x_{\pi(1)}, ..., x_{\pi(n)})$$

Example 1 *The n-permutations of $f_{110} = x_1x_2x_3 + x_2x_3 + x_2 + x_3$ are*

$$f_{110}^{\pi_0} = f_{110}^{\pi_1} = x_1x_2x_3 + x_2x_3 + x_2 + x_3.$$
$$f_{110}^{\pi_2} = f_{110}^{\pi_4} = x_1x_2x_3 + x_1x_3 + x_1 + x_3 = f_{122}.$$
$$f_{110}^{\pi_3} = f_{110}^{\pi_5} = x_1x_2x_3 + x_1x_2 + x_1 + x_2 = f_{124}.$$

2. **Definition 5** ***[q-permutation of f]*** *For an argument x of f which takes a value from Q, define a permutation $\varphi \in S_q$ as a bijection $x^{\varphi} : Q \to Q$. Then consider a list of permutations $\varphi(n) = (\varphi_1, ..., \varphi_n)$ where $\varphi_i \in S_q,\ 1 \leq i \leq n$ or $\varphi(n) \in S_q^n = S_q \times \cdots \times S_q$ (direct product of n copies of S_q) . Now we define the* q-permutation *of f by*

$$f\varphi(n)(\mathbf{x}_n) = f(x_1^{\varphi_1}, ..., x_n^{\varphi_n}). \tag{6}$$

Example 2 *For the binary case $Q = \{0, 1\}$ the permutations $\varphi(n)$ is expressed by a binary word $\varphi(a_1 \cdots a_n)$ which* operates *on $\mathbf{x}_n$ such that $x_i^{a_i} = x_i$ if $a_i = 0$ and $x_i^{a_i} = x_i + 1$ if $a_i = 1$ (Boolean negation). For example $f_{110}\varphi(100) = (x_1+1)x_2x_3+x_2x_3+x_2+x_3 = x_1x_2x_3+x_2+x_3 = f_{230}$, $f_{110}\varphi(110) = f_{185}$ and so on. In general for a prime number of states $Q = \{0, 1, ..., p-1\} = GF(p)$, the permutation of Q is expressed by an addition modulo p such that $x + a, a \in Q$.*

3. **Definition 6** ***[(n, q)-permutation of f]*** *Combining n-permutation and q-permutation with an additional permutation $\psi : Q \to Q$ of the function value, we finally define a* unified *permutation of f called* (n, q)-permutation *of f which is expressed by a 3-tuple of permutations $(\pi, \psi, \varphi(n))$.*

$$(\pi, \psi, \varphi(n))f(\mathbf{x}_n) = \psi f^{\pi}\varphi(n)(\mathbf{x}_n) = \psi f(x_{\pi(1)}^{\varphi_1}, ..., x_{\pi(n)}^{\varphi_n}). \tag{7}$$

Example 3

$$\begin{aligned}(\pi_2, (1, 2), \varphi(100))f_{110} &= f_{110}(x_2^1, x_1^0, x_3^0) + 1 \\ &= (x_2 + 1)x_1x_3 + x_1x_3 + x_1 + x_3 + 1 \\ &= x_1x_2x_3 + x_1 + x_3 + 1 \\ &= f_{37}.\end{aligned}$$

All (n, q)-permutations of f_{110} are given in Example 5.

4 Generalized automorphism of CA

In this section, using the (n,q)-permutation of f, we define a generalized automorphism called *g-automorphism* of CA and prove that the set of the g-automorphisms constitutes a group under a rule of the semi-direct product.

Definition 7 *For two CA $A = (f,\nu)$ and $A' = (f',\nu')$, A is called* g-automorphic *with A' denoted $A \cong_g A'$, if and only if there is an (n,q)-permutation $(\pi,\psi,\varphi(n))$ such that the following equation holds.*

$$(f',\nu') = (\psi f^{\pi}\varphi(n), \nu^{\pi}). \tag{8}$$

Remarks 1 *If for any $\varphi \in S_q, \varphi_i = \varphi,\ 1 \le i \le n$, then by taking $\psi = \varphi^{-1}$, g-automorphism becomes the original automorphism (5; 6).*

We show here that the set of the 3-tuples of permutations

$$G_{n,q} = \{(\pi,\psi,\varphi(n)) \mid \pi \in S_n, \psi \in S_q, \varphi(n) \in S_q^n\}$$

is a group. The order of $G_{n,q}$ is $n!q^{n+1}$.

Theorem 2 *Let $g = (\pi,\psi,\varphi(n)) \in G_{n,q}$ and $g' = (\pi',\psi',\varphi'(n)) \in G_{n,q}$. Then $G_{n,q}$ is a group under the rule of* semi-direct product*;*

$$g'g = (\pi',\psi',\varphi'(n))(\pi,\psi,\varphi(n)) = (\pi'\pi, \psi'\psi, \varphi'(n)^{\pi}\varphi(n)), \tag{9}$$

where $\varphi'(n)^{\pi}\varphi(n) = (\varphi'_{\pi(1)}\varphi_1, ..., \varphi'_{\pi(n)}\varphi_n)$ is the componentwise group operation of the direct product S_q^n.

Proof: The proof is done in the same way as the proof given by M. Harrison for Boolean functions, see page 822 of (2). He utilizes Theorem 6.5.1, page 88, Section 6.5 of the text book by M. Hall (1), where the semi-direct product $K \rtimes_{\varphi} H$ of K by H is defined by the rule

$$[h_1,k_1]\cdot[h_2,k_2] = [h_1h_2, k_1^{h_2}k_2], \tag{10}$$

where $h_1,h_2 \in H, k_1,k_2 \in K$ and the automorphism φ^2 of K is defined by for any $h \in H, k \rightleftharpoons k^h$ for all $k \in K$. The product rule (10) is shown well defined: (1) associative, (2) the identity is $[\mathbf{1},\mathbf{1}]$ and (3) a left inverse $[h,k]^{-1}$ of $[h,k]$ is $[h^{-1},(k^{-1})^{h^{-1}}]$.

At applying this standard rule of the semi-direct product to the 3-tuples in Equation (9), first consider the semi-direct product $S_n \rtimes_{\varphi} S_q^n$ and then combine $\psi \in S_q$ as a direct product. □

The following example will help understanding the semi-direct product of $G_{n,q}$.

Example 4 *Suppose that two group elements $g_1 = (\pi_1,\psi_0,\varphi(100))$ and $g_2 = (\pi_2,\psi_0,\varphi(001))$ in $G_{3,2}$ act[3] on $f_{110} \in \mathcal{P}_{3,2}$ in this order where $\psi_0 = \mathbf{1}$. That is*

$$\begin{aligned} g_1 \circ f_{110} &= x_1x_2x_3 + x_2 + x_3 = f_{230} \\ g_2 \circ (g_1 \circ f_{110}) &= g_2 \circ f_{230} = x_1x_2x_3 + x_1x_2 + x_1 + x_3 + 1 = f_{229}. \end{aligned}$$

[2] Note that this symbol φ is independent from our permutation φ.

[3] The symbol of group action $\circ$ is usually omitted like group operation.

On the other hand, by applying the rule of the semi-direct product (9), we see

$$\begin{aligned} g_2 g_1 &= (\pi_2, \psi_0, \varphi(001))(\pi_1, \psi_0, \varphi(100)) \\ &= (\pi_2\pi_1, \psi_0, \varphi(001)^{\pi_1}\varphi(100)) \\ &= (\pi_2\pi_1, \psi_0, \varphi(010)\varphi(100)) \\ &= (\pi_3, \psi_0, \varphi(110)) \\ &= g_3 \end{aligned}$$

But

$$g_3 \circ f_{110} = x_1x_2x_3 + x_1x_2 + x_1 + x_3 + 1 = f_{229}.$$

Lemma 3 *Any g-automorphic CA are equivalent (have the same global function) up to permutation.*

Proof: It is obvious from Equation (8). Permute the local function f with the inverses of $\varphi(n)$ and ψ. □

Example 5 *[g-automorphism class of f_{110}] As a typical example of g-automorphism classification, we consider f_{110} again. Table 1 below lists up the (n, q)-permutations of f_{110} only for the case of $\psi_0 = \mathbf{1}$. The permutation $\psi_1 f^{\pi}\varphi$ where $\psi_1 = (12)$ is obtained by adding 1 to the polynomial of each entry. For example for $\psi_0 f^{\pi_2}\varphi(010) = f_{167} = x_1x_2x_3 + x_1x_3 + x_2x_3 + x_1 + 1$, we have $\psi_1 f^{\pi_2}\varphi(010) = x_1x_2x_3 + x_1x_3 + x_2x_3 + x_1 = f_{88}$.*

Tab. 1: g-automorphism class of f_{110}

$\psi, \varphi \backslash \pi$	π_0	π_1	π_2	π_3	π_4	π_5
$\psi_0 f\varphi(000)$	110	110	122	122	124	124
$\psi_0 f\varphi(100)$	230	230	218	218	188	188
$\psi_0 f\varphi(010)$	155	157	167	181	199	211
$\psi_0 f\varphi(001)$	157	155	181	167	211	199
$\psi_0 f\varphi(110)$	185	217	173	229	203	227
$\psi_0 f\varphi(101)$	217	185	229	173	203	203
$\psi_0 f\varphi(011)$	103	103	91	91	103	61
$\psi_0 f\varphi(111)$	118	118	94	94	62	62

For $f \neq f' \in \mathcal{P}_{3,2}$, it is seen that $\psi_1 f \neq f$ and $\psi_1 f \neq \psi_1 f'$. Since Table 1 contains 24 different functions among the $3!2^3 = 48$ entries, it is seen that the number of the functions that are g-automorphic with f_{110} is $24 \times 2 = 48$. Then by Lemma 3, we see

Lemma 4 *There are 48 local functions which are computation universal up to permutation.*

This is compared with 6 functions which are automorphic with f_{110} (5; 6).

5 Generalized automorphism classification of CA

g-automorphism $\cong_g$ is an equivalence relation in $\mathcal{P}_{n,q}$ and naturally induces a generalized classification of CA called *g-automorphism classification*. Every local function in a class has the same global property *up to permutation* by Lemma 3.

5.1 g-automorphism classification of ELF

The classification of 256 ELFs into 11 g-automorphism classes (denoted GN class) is shown in Table 2, where every g-automophism classes a union of NW classes. The NW classification table will be found in H. Nishio (2009) (5). 6 functions in GN6** are reversible and 32 functions in GN9*, GN10* and GN11* are surjective but not injective. The rests are not surjective nor injective. GN8 consists of 48 universal functions.

Tab. 2: g-automorphism classification of 2-state 3-neighbor CA

GN class	size	NW classes
$GN1$	2	$NW1$
$GN2$	44	$NW2 \cup NW6 \cup NW10 \cup NW22 \cup NW38 \cup NW43$
$GN3$	22	$NW3 \cup NW7 \cup NW11 \cup NW29 \cup NW34$
$GN4$	24	$NW4 \cup NW9 \cup NW37$
$GN5$	24	$NW5 \cup NW8 \cup NW20 \cup NW35$
$GN6**$	6	$NW12** \cup NW44**(reversible)$
$GN7$	54	$NW13 \cup NW14 \cup NW15 \cup NW18 \cup NW21 \cup NW23 \cup NW26$ $\cup NW33 \cup NW36 \cup NW39 \cup NW45 \cup NW46$
$GN8$	48	$NW16 \cup NW17 \cup NW24 \cup NW28 \cup NW32 \cup NW41$ $(universal)$
$GN9*$	24	$NW19* \cup NW25* \cup NW31* \cup NW42*$
$GN10*$	6	$NW27*$
$GN11*$	2	$NW30* \cup NW40*$
total	256	46 NW classes

5.2 Counting problem

From the group theory point of view, g-automorphism classification is considered as a group action of $G_{n,q}$ on $\mathcal{P}_{n,q}$ and, for instance, the number of g-automorphism classes will be computed by use of the Pólya's counting theory (8) as D. Slepian and M. Harrison did. Computing the cycle index of the permutation group $G_{n,q}$ which acts on $\mathcal{P}_{n,q}$ seems a new problem. It will certainly reflect the symmetric structure of the polynomials over finite field. We have, however, not obtained it yet.

6 Concluding remarks and acknowledgments

We have generalized the automorphism (classification) of CA by considering two kinds of permutations of the local structures; n-permutation of the neighborhood and q-permutation of the cell states. For explaining the idea, we inserted several examples using rule f_{110} and gave the table of g-automorphisms

of f_{110}. As a byproduct we see that 48 local rules are universal up to permutation. We also gave the g-automorphism classification of 256 ELF into 11 g-automorphism classes. The counting problem of the number of the g-automorphism classes has been left for future research.

The author thanks Thomas Worsch from the University of Karlsruhe for establishing Theorem 1 and his student Clemens Lode for his Java program **catest106d**. He also thanks Mitsuhiko Fujio from the Kyushu University of Technology and Fumihiro Ushitaki from the Kyoto Sangyo University for having interest in this work from the point of view of the action of permutation groups.

References

[1] Hall, M.: *The Theory of Groups*, The Macmillan Company, 1959.

[2] Harrison, M. A.: The Number of Transitivity sets of Boolean Functions, *J. Soc. Indust. Appl. Math.*, **11**, 1963, 806–828.

[3] Lode, C.: http://www.clawsoftware.com/projects/catest/.

[4] Nishio, H.: Changing the Neighborhood of Cellular Automata, *Proceedings of MCU2007, eds. J. Durand-Lose and M. Margenstern*, LNCS 4664, 2007.

[5] Nishio, H.: AUTOMORPHISM CLASSIFICATION OF CELLULAR AUTOMATA, *Proceedings of Workshop on Non-Classical Models for Automata and Applications(NCMA), books@ocg.at*, 2009.

[6] Nishio, H.: Automorphism Classification of Cellular Automata, 2010, Submitted to Fundamenta Informaticae, Special Issue on Non-Classical Models for Automata and Applications(NCMA).

[7] Nishio, H., Worsch, T.: Changing the neighborhood of cellular automata : local structure, equivalence and isomorphism., *J. Cellular Automata*, **5**(3), 2010, 227–240.

[8] Pólya, G., Read, R. C.: *Combinatorial Enumeration of Groups, Graphs and Chemical Compounds*, Springer-Verlag, 1987.

[9] Shannon, C. E.: The Synthesis of Two-Terminal Switching Circuits, *Bell Systems Tech. J.*, **28**, 1949, 59–98.

[10] Slepian, D.: On the number of symmetry types of Boolean functions of n variables, *Canadian J. Math.*, **5**, 1953, 185–193.

[11] Worsch, T., Nishio, H.: Achieving universality of CA by changing the neighborhood, *J. Cellular Automata*, **4**(3), 2009, 237–246.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 287–298

Dynamical Properties of Rule 56 Elementary Cellular Automaton of Wolfram Class II

Fumio Ohi

Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya 466-8555, Japan, E-mail: ohi.fumio@nitech.ac.jp

Rule 56 elementary cellular automaton belongs to Wolfram class II and shows us simple right shift space-time patterns for randomly given initial configurations by computer simulation. But precisely examining the dynamics of rule 56, we have unexpected patterns. In this paper we examine the dynamical properties of the rule in detail and shows that the rule has three chaos dynamical sub-systems, two of which are subshifts of finite type and generate right or left shift patterns, but space-time patterns generated by other one are neither right nor left shift patterns. All of these three dynamical systems are Devaney chaos.

Keywords: rule 56, elementary cellular automata, Devaney chaos

1 Introduction

Wolfram's classification of cellular automata based on an extensive computer simulation is well known and the space-time patterns generated by members of the class II are simple and it is said that there are different possible final states, but they consist of a certain set of simple configurations that either remain the same forever or repeat every few time steps. See Wolfram [6]. Rule 56 elementary cellular automaton(ECA) is a member of the class II, and it is observed by computer simulation that the rule generates right sift patterns for randomly given initial configurations.

In this paper we examine the dynamical properties of rule 56 in detail and show that the global transition function of the rule has three chaos dynamical sub-systems, two of which are subshifts of finite type and generate right or left shift patterns, but space-time patterns generated by other one are neither right nor left shift patterns. In the process of the examination, it is shown that there is a configuration of which time development varies every time step and does not settle down. These three dynamical sub-systems are Devaney chaos.

An ECA is defined to be a tuple $(\{0,1\}, g)$, where g is a mapping from $\{0,1\}^3$ to $\{0,1\}$ and is called a local transition function. An ECA is determined by g and is simply called an ECA g. There exist $2^8 = 256$ ECA's and each of them has the rule number defined by $\sum_{a,b,c} g(a,b,c)2^{a2^2+b2+c}$. We denote the local transition function having rule number r as g_r.

An ECA g defines a mapping $\boldsymbol{g}$ from $\mathcal{A} \equiv \{0,1\}^{\mathbb{Z}}$ to $\mathcal{A}$, which is called the global transition function of the ECA, as

$$\boldsymbol{x} = (\cdots, x_{-1}, x_0, x_1, \cdots) \in \mathcal{A}, \quad (\boldsymbol{g}(\boldsymbol{x}))_i = g(x_{i-1}, x_i, x_{i+1}), \quad i \in \mathbb{Z}.$$

We usually use the bold face of the letter denoting the global transition function for the corresponding local transition function. An element of $\mathcal{A}$ is called a configuration.

The left and right shift transformations are written as $\sigma_L : \mathcal{A} \to \mathcal{A}$ and $\sigma_R : \mathcal{A} \to \mathcal{A}$, respectively.

Defining a metric d on $\mathcal{A}$ as $d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=-\infty}^{\infty} \frac{|x_i - y_i|}{2^{|i|}}$ for $\boldsymbol{x},\ \boldsymbol{y} \in \mathcal{A}$, we have a topological dynamical system $(\mathcal{A}, \boldsymbol{g})$, which defines an orbit of an arbitrarily given initial configuration $\boldsymbol{x} \in \mathcal{A}$ as $\boldsymbol{g}^0(\boldsymbol{x}) = \boldsymbol{x},\ \boldsymbol{g}^{t+1}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{g}^t(\boldsymbol{x})),\ t \geq 0$. A topological dynamical system $(\mathcal{S}, \boldsymbol{g})$ is a sub-system of $(\mathcal{A}, \boldsymbol{g})$ if $\mathcal{S} \subseteq \mathcal{A}$ and $\boldsymbol{g}(\mathcal{S}) \subseteq \mathcal{S}$. The metric on $\mathcal{S}$ is the restriction of the metric d. A topological dynamical system is called Devaney chaos when it has a dense orbit and the class of all periodic configurations is dense in $\mathcal{S}$. See G.Cattaneo, et al. [1].

The local transition function of rule 56 is given along with that of rule 40 of Wolfram class I in the following table. In F. Ohi [3] it is shown that $(\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{40})$ is a right-shift dynamical system and $\lim_{t\to\infty} \boldsymbol{g}^t(\boldsymbol{x}) = \mathbf{0}$ for $\boldsymbol{x} \in \mathcal{A} \backslash \mathcal{S}_{0(1),1(1,2)}$. (The definitions of terminologies are given in Section 1.1.) From the table we may think that these two rules are close and the dynamical structure of rule 56 is almost same as rule 40.

In this paper we show that $(\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{56}) = (\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{40}) = (\mathcal{S}_{0(1),1(1,2)}, \sigma_L)$ and $(\mathcal{S}_{1(1)}, \boldsymbol{g}_{56}) = (\mathcal{S}_{1(1)}, \sigma_R)$ are subshifts of finite type(SFT's). These two SFT's are Devaney chaos and topologically mixing, since the transition matrix of each SFT is irreducible and aperiodic. Symbolic dynamical approach to rule 56 is given in the section 2.1.

(a,b,c)	$(1,1,1)$	$(1,1,0)$	$(1,0,1)$	$(1,0,0)$	$(0,1,1)$	$(0,1,0)$	$(0,0,1)$	$(0,0,0)$
$g_{40}(a,b,c)$	0	0	1	0	1	0	0	0
$g_{56}(a,b,c)$	0	0	1	1	1	0	0	0

Furthermore Rule 56 has another chaotic dynamical sub-system $(\mathcal{Y}, \boldsymbol{g}_{56})$, where $\mathcal{Y}$ is a set of configurations of special type defined in section 1.1 and a subset of $\mathcal{S}_{1(1,2)} \setminus \left(\mathcal{S}_{0(1),1(1,2)} \cup \mathcal{S}_{1(1)}\right)$. Since $\mathbf{g}_{56}$ is neither right nor left shift on $\mathcal{Y}$, we have some difficulties for examination of the dynamical properties from the symbolic point of view as SFT's or sofic systems. Our examination is straightforward.

The space-time pattern of a configuration of $\mathcal{Y}$ shows us a kind of confliction between left and right groups of which front line moves like a wave and neither remains the same configuration forever nor repeats some specific configurations. This movement has not been found out by computer simulation with randomly given initial configurations and is not be given by rule 40.

The correspondence $\binom{100}{1}$ of rule 56, which is only one different point from $\binom{100}{0}$ of rule 40, makes the dynamical properties of rule 56 richer than rule 40. This correspondence emerges intrinsic interactions between 0 and 01, between 0 and 011 and between 01 and 011, which make the wave-like motions for configurations of $\mathcal{Y}$. We explain the interactions in Section2.2. In Section 3, Devaney chaos property of $(\mathcal{Y}, \mathbf{g}_{56})$ is proved by practically constructing transitive and periodic configurations and in the process of the construction the interactions are consistently used.

1.1 Notations

For our straightforward examination, we need the following notations.

(1) For $\boldsymbol{\alpha}_i \in \{0,1\}^{n_i},\ \boldsymbol{\beta}_i \in \{0,1\}^{m_i},\ n_i \geq 1,\ m_i \geq 1,\ i \in \mathbb{Z}$, we define

$$(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)_{i=-\infty}^{+\infty} = (\cdots, \alpha_1^{-1}, \cdots, \alpha_{n_{-1}}^{-1}, \beta_1^{-1}, \cdots, \beta_{m_{-1}}^{-1},$$
$$\alpha_1^0, \cdots, \alpha_{n_0}^0, \beta_1^0, \cdots, \beta_{m_0}^0, \alpha_1^1, \cdots, \alpha_{n_1}^1, \beta_1^1, \cdots, \beta_{m_1}^1, \cdots),$$

where $\boldsymbol{\alpha}_i = (\alpha_1^i, \cdots, \alpha_{n_i}^i),\ \boldsymbol{\beta}_i = (\beta_1^i, \cdots, \beta_{m_i}^i),\ i \in \mathbb{Z}$.

For a pair $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, $\boldsymbol{\alpha} \in \{0,1\}^n,\ \boldsymbol{\beta} \in \{0,1\}^m$

$$(\boldsymbol{\alpha}, \boldsymbol{\beta})_k = (\underbrace{\boldsymbol{\alpha}, \boldsymbol{\beta}, \cdots, \boldsymbol{\alpha}, \boldsymbol{\beta}}_{k \text{ pairs}}),\ k \in \mathbb{N} \cup \{0\},$$

which means empty, when $k = 0$.

(2) $0_k = (\underbrace{0, \cdots, 0}_{k}),\ 1_k = (\underbrace{1, \cdots, 1}_{k}),\ k \in \mathbb{N} \cup \{0\}$.

(3) We intensively use the following terminology.

$$\mathcal{S}_{0(m_1,\cdots,m_p),1(n_1,\cdots,n_q)} = \{(\mathbf{0}_{i_j}, \mathbf{1}_{k_j})_{j=-\infty}^{\infty} \mid i_j = m_1 \text{ or} \cdots \text{or } m_p, k_j = n_1 \text{ or} \cdots \text{or } n_q\}\ ,$$
$$\mathcal{S}_{1(n_1,\cdots,n_q)} = \{(\mathbf{0}_{i_j}, \mathbf{1}_{k_j})_{j=-\infty}^{\infty} \mid i_j \in \mathbb{N}, k_j = n_1 \text{ or} \cdots \text{or } n_q\}\ .$$

(4) $\mathcal{A} = \{0,1\}^{\mathbb{Z}}$ and $\mathcal{X} = \mathcal{S}_{1(0,1,2)} \backslash \left(\mathcal{S}_{0(1),1(1,2)} \cup \mathcal{S}_{1(1)}\right)$. The subset $\mathcal{Y} \subseteq \mathcal{X}$ is defined to be the set of configurations of the following type,

$$\cdots (01)_{k_i} 0_{l_i} \cdots (01)_{k_1} 0_{l_1} 011 (01)_{n_1} 011 (01)_{n_2} \cdots 011 (01)_{n_{l_1}} \cdots\ ,$$

where $n_i \geq 0,\ l_i \geq 1,\ k_i \geq 1$. It is easy to see that

$$\mathcal{S}_{0(1),1(1,2)} \cap \mathcal{S}_{1(1)} = \{(\cdots, 0, 1, 0, 1, \overset{0}{\overbrace{0}}, 1, 0, 1, 0, \cdots), (\cdots, 0, 1, 0, 1, 0, \overset{0}{\overbrace{1}}, 0, 1, 0, \cdots)\}$$

2 Basic Properties of Rule 56

2.1 Dynamics of $\boldsymbol{g}_{56}$ on $\mathcal{S}_{0(1),1(1,2)}$ and $\mathcal{S}_{1(1)}$ - Symbolic dynamical approach -

We follow B.P.Kitchens [2] for the terminologies of symbolic dynamics.

Theorem 2.1. $\boldsymbol{g}_{56}(\mathcal{A}) \subseteq \mathcal{S}_{1(0,1,2)},\quad \boldsymbol{g}_{56}\left(\mathcal{S}_{1(1,2)}\right) \subseteq \mathcal{S}_{1(1,2)},\quad \boldsymbol{g}_{56}(\{\mathbf{0}\}, \{\mathbf{1}\}) = \{\mathbf{0}\}$

Proof: Noticing

$$\binom{1,1,1}{0},\quad \binom{1,1,0}{0},\quad \binom{0,1,1}{1},\quad \binom{0,1,0}{0},\quad \binom{1,0,1}{1}$$

in the table of the local transition function of rule 56, we have the one step transition

		← three or more 1's →							
$\cdots$	0	1	1	$\cdots$	1	1	1	0	$\cdots$
$\cdots$	$*_1$	1	0	$\cdots$	0	0	0	$*$	$\cdots$

For $*_1$ to be 1, the above transition should be

			← three or more 1's →							
$\cdots$	1	0	1	1	$\cdots$	1	1	1	0	$\cdots$
$\cdots$	$*_2$	1	1	0	$\cdots$	0	0	0	$*$	$\cdots$

and in this case $*_2$ is necessarily 0 from the local transition function of rule 56. Then we have two possibilities

$$\begin{array}{ccccccccccc} & & \multicolumn{6}{c}{\leftarrow \text{three or more 1's} \rightarrow} & & & \\ \cdots & 0 & 1 & 1 & \cdots & 1 & 1 & 1 & 0 & \cdots \\ \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & * & \cdots \end{array} \qquad \begin{array}{ccccccccccc} & & & & & \multicolumn{6}{c}{\leftarrow \text{three or more 1's} \rightarrow} & & \\ \cdots & * & 0 & 1 & 1 & \cdots & 1 & 1 & 1 & 0 & \cdots \\ \cdots & 0 & 1 & 1 & 0 & \cdots & 0 & 0 & 0 & * & \cdots \end{array}$$

which means that a 1-block of three or more length is reduced to the one of one or two length in one time step, where $* = 0$ or 1 according to $g_{56}(100) = 0$ or $g_{56}(101) = 1$, respectively.

For a 1-block of three or more length to emerge in time-development we need (a, b, c, d, e) such that $\boldsymbol{g}_{56}(a, b, c, d, e) = 111$, which is equivalent to $g_{56}(a, b, c) = 1,\ g_{56}(b, c, d) = 1,\ g_{56}(c, d, e) = 1$, but we cannot construct such (a, b, c, d, e) from the table of the local transition function of rule 56.

From Theorem 2.1, $\boldsymbol{g}_{56}(\boldsymbol{x}) \in \mathcal{S}_{1(1,2)}$ for every $\boldsymbol{x} \in \mathcal{A}$, then the dynamical system $(\mathcal{S}_{1(1,2)}, \boldsymbol{g}_{56})$ is essentially $(\mathcal{A}, \boldsymbol{g}_{56})$.

It is easily shown from the local transition function of rule 56 that $(\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{56}) = (\mathcal{S}_{0(1),1(1,2)}, \sigma_L)$. $\mathcal{S}_{0(1),1(1,2)}$ is determined by a set of words $\mathcal{W} = \{(010), (011), (101), (110)\}$, i.e.,

$$\mathcal{S}_{0(1),1(1,2)} = \{\boldsymbol{x} \in \mathcal{A} \mid \forall i \in \mathbb{Z}, (x_i, x_{i+1}, x_{i+2}) \in \mathcal{W}\}$$

Letting a transition matrix A be

$$A = \begin{array}{c} \\ (010) \\ (011) \\ (101) \\ (110) \end{array} \begin{array}{c} \begin{array}{cccc} (010) & (011) & (101) & (110) \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array},$$

the left-subshift of finite type (Σ_A, σ_L) determined by the transition matrix is conjugate to $(\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{56})$. Noticing that the transition matrix A is irreducible and aperiodic, we have the following theorem.

Theorem. 2.2. $\big(\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{56}\big) = \big(\mathcal{S}_{0(1),1(1,2)}, \sigma_L\big)$ and $\big(\mathcal{S}_{1(1)}, \boldsymbol{g}_{56}\big) = \big(\mathcal{S}_{1(1)}, \sigma_R\big)$ hold, and both of them are Devaney chaos and topologically mixing.

The proposition about $(\mathcal{S}_{1(1)}, \boldsymbol{g}_{56})$ is also easy by noticing that $\mathcal{S}_{1(1)}$ is determined a set of words $\mathcal{V} = \{(00), (01), (10)\}$ and the corresponding transition matrix is given by

$$\begin{array}{c} \\ (00) \\ (01) \\ (10) \end{array} \begin{array}{c} \begin{array}{ccc} (00) & (01) & (10) \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array},$$

which is also irreducible and aperiodic.

The dynamical system $(\mathcal{Y}, \mathbf{g}_{56})$, which is examined in the sequel of this paper, is neither right nor left subshift of $(\mathcal{A}, \mathbf{g}_{56})$, and then not a sofic system even when the attractors $\mathcal{S}_{1(1)}$ and $\mathcal{S}_{0(1),1(1,2)}$ are included. We practically construct periodic and transitive configurations of $\mathcal{Y}$ to prove the chaotic property of the dynamical system.

2.2 Basic dynamical properties of configurations of $\mathcal{X}$

In the sequel we write 01 to denote a block 01 not a part of the block 011. NB-0 (non-belonging 0) denotes a 0 which does not a member of 01 and 011.

Figures 1 and 2 show us key dynamical properties of rule 56. Black and white circle in the figures mean 1 and 0 state, respectively.

From Fig. 1, we see that $0_n(011)_m$ changes to $0_{n-1}01(011)_{m-1}$ and then to $0_{n-1}(011)_{m-1}$. This two-step transition tells us that NB-0 changes 011 to 01 and is erased, and NB-0 erases 01. Fig. 2 shows us that $(01)_n(011)_m$ changes to $(01)_{n-1}(011)_m$ in one time step, which shows us that 01 is erased by the right 011. These three interactions between NB-0 and 011, NB-0 and 01, 01 and 011 are crucial for the time development of configurations of $\mathcal{Y}$.

By these interactions, we have an example of time development shown in Fig.3.

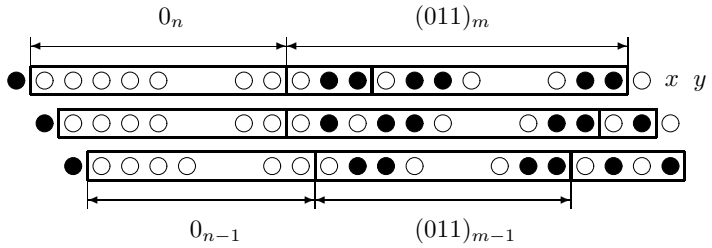


Fig. 1: Interactions between NB-0 and 011, and between NB-0 and 01. $xy = 10$ or 01 or 00.

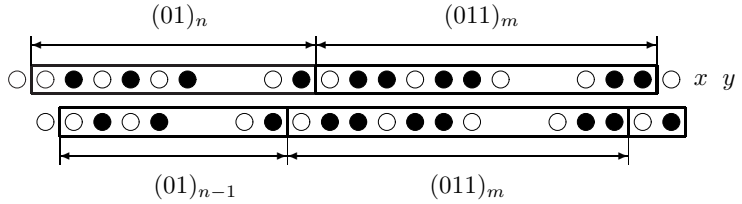


Fig. 2: Interaction between 01 and 011. $xy = 10$ or 01 or 00.

From Fig.3 we know that a block 011 basically moves leftward and changes to 01, when coming upon a NB-0, and this NB-0 is erased. A block 01 is erased by a left-hand NB-0 and this NB-0 is not erased.

Utilizing the interactive properties, we can easily construct a periodic configuration of $\mathcal{Y}$. We examine the time-development of the following configuration.

$$
\begin{array}{cccccccc}
 & & \text{the left group} & \rightarrow & & \leftarrow & \text{the right group} & \\
 & \big| \text{ the second block } \big| & \text{ the first block } & \big| & \text{ the first block } & \big| & \text{ the second block } & \big| \\
\boldsymbol{a} \equiv & \cdots & (01)_{\sum_{i=1}^{k-1} n_i + k} 0_k & (01)_{\sum_{i=1}^{k-1} n_i + k} 0_k & (011(01)_{n_i})_{i=1}^{k-1} 011 & (011(01)_{n_i})_{i=1}^{k-1} 011 & \cdots &
\end{array}
$$

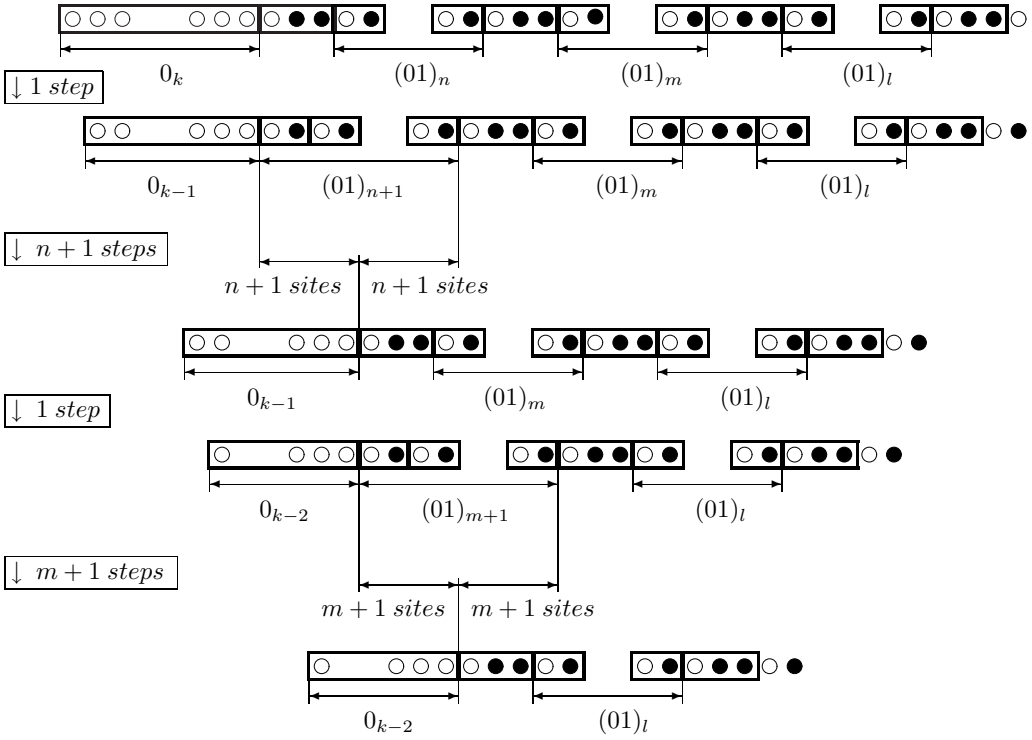


Fig. 3: Typical dynamical properties of the blocks 011, 01 and NB-0, and their interactions. A block 011 moves leftward by $n+1$ sites, when there exist $n+1$ NB-0's on the left hand, and a NB-0 moves rightward by $m+1$ sites, when there exist $m+1$ 01's on the right hand. And a block 011 moves leftward by $m+1$ sites, when there exist $m+1$ 01's on the left hand.

Referring to Fig.4, we can observe; 0_k of the first block of the left group erases the first block of the right group and the front line moves to the right by $\sum_{i=1}^{k-1} n_i + k$ sites in $\sum_{i=1}^{k-1} n_i + 2k$ time steps. Then the second $(011(01)_{n_i})_{i=1}^{k-1}$ 011 moves to the left by $\sum_{i=1}^{k-1} n_i + k$ sites, erasing the block $(01)_{\sum_{i=1}^{k-1} n_i + k}$ and we have the configuration $\boldsymbol{a}$ having the period $2\sum_{i=1}^{k-1} n_i + 3k$.

Setting the values of the parameters $k,\ n_i (i \geq 1)$ of the configuration $\boldsymbol{a}$ appropriately, we have the following theorem about the periodic properties of rule 56.

Theorem.2.3. $\boldsymbol{g}_{56}$ has a period- p configuration in $\mathcal{S}_{1(1,2)} \backslash \left(\mathcal{S}_{0(1),1(1,2)} \cup \mathcal{S}_{1(1)} \right)$, where $p = 3, 6, 8, 9, 10, 11, 12, 13, \cdots$

Proof: For any even integer $p \geq 8$, $p = 2m = 3 \cdot 2 + 2 \cdot (m-3),\ m \geq 4$ holds. Setting $k = 2,\ n_1 = m-3$, we have a configuration $\boldsymbol{a}$ with period p. For any odd integer $p \geq 9$, $p = 3 \cdot 3 + (p-9)$, $p-9$ is an even number and can be written as $2 \cdot q,\ q \geq 0$. Then p is easily expressed as $p = 2(n_1 + n_2) + 3k$ by

setting the parameters $n_1 = q,\ n_2 = 0$ and $k = 3$.

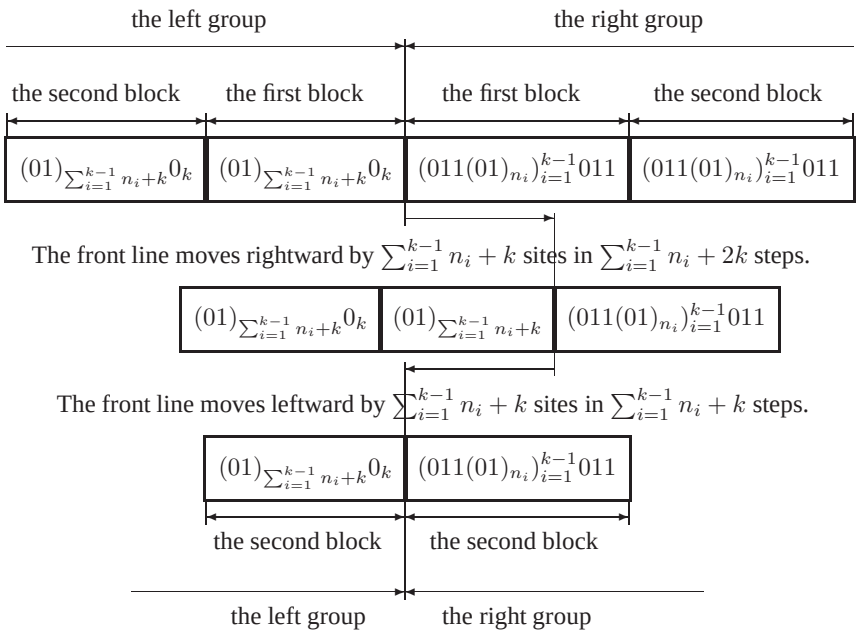


Fig. 4: General form of periodic configuration and its time development.

The following examples are periodic configurations having period 2, 5 and 7, respectively.

$$\cdots 01010101 \cdots,$$
$$\cdots 010110101101011 \cdots,$$
$$\cdots 010101101010110101011 \cdots,$$

which are members of $\mathcal{S}_{0(1),1(1,2)}$ and not of $\mathcal{S}_{1(1,2)} \setminus \left(\mathcal{S}_{0(1),1(1,2)} \cup \mathcal{S}_{1(1)}\right)$.

It has been shown that the dynamical system $(\mathcal{A}, \boldsymbol{g}_{56})$ has every periodic configuration except for period 4. It remains to be an open problem to make sure whether there exists a period-4 configuration.

3 Dynamics on $\mathcal{X}$

The one-stage time development of a configuration of $\mathcal{Y}$

$$\cdots (01)_{k_i}0_{l_i} \cdots (01)_{k_1}0_{l_1}011(01)_{n_1}011(01)_{n_2} \cdots 011(01)_{n_{l_1}} \cdots,$$

where $n_i \geq 0,\ l_i \geq 1,\ k_i \geq 1$, is shown in Fig.5 is basic for the examination of the chaotic property of the rule 56 on $\mathcal{Y}$.

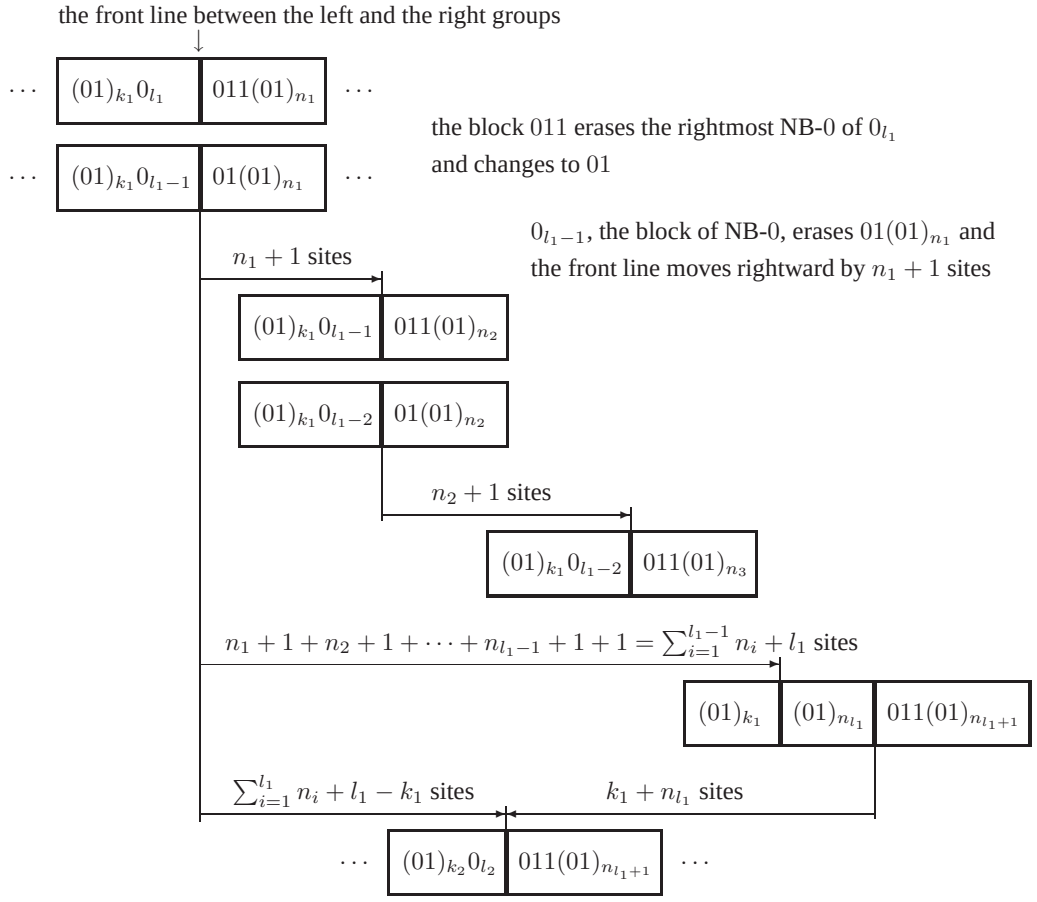


Fig. 5: One-stage time development of a configuration of $\mathcal{Y}$. The block $(01)_{k_1}0_{l_1}$ is erased by $011(01)_{n_1}\cdots011(01)_{n_{l_1}}$, and the front line moves rightward by $\sum_{i=1}^{l_1}n_i+l_1-k_1$ sites. When this quantity is negative, the front line consequently arrives at a left side of the original position at time 0.

The moving distance of the front line after n stages is

$$A_n \equiv \sum_{i=1}^{l_1+\cdots+l_n}(n_i+1)-\sum_{i=1}^{n}k_i,$$

and thus when the co-ordinate number of the right site of the front line is initially n_0, the co-ordinate number of the cell is n_0+A_n after the $n-$th stage . It is easily imagined that the trajectory of $\{n_0+A_n\}_{n\geq 0}$ may form a wave like form, depending on values of the parameters $k_i, l_i, n_i, i\geq 1$. We have the following

Theorem.

Theorem.3.1. $(\mathcal{Y}, \boldsymbol{g}_{56})$ is Devaney chaos.

Proof: The set of periodic configurations is dense in $\mathcal{Y}$.

For a configuration $\boldsymbol{x} \in \mathcal{Y}$, we assume the pattern as the following. We take a finite sub-configuration around the origin of the configuration as shown in Fig.6, not changing the co-ordinate number 0 site.

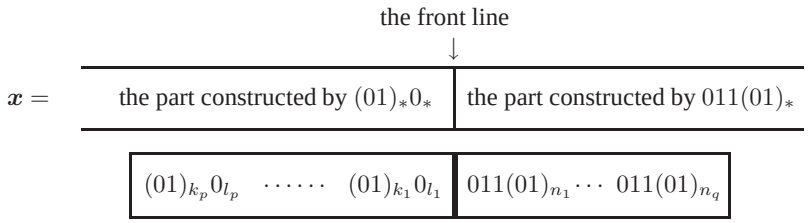


Fig. 6: A finite sub-configuration located around the origin of the configuration $\boldsymbol{x}$.

The numbers of NB-0's and 011's in the sub-configuration are $\sum_{i=1}^{p} l_i$ and q, respectively.

(i) When $\sum_{i=1}^{p} l_i \geq q$, then we add $\sum_{i=1}^{p} l_i - q$ 011's to the right side of the sub-configuration.

(ii) When $\sum_{i=1}^{p} l_i \leq q$, then we add $q - \sum_{i=1}^{p} l_i$ NB-0's to the left side of the sub-configuration.

The proof is logically same in either case, then we assume (i) without loss of generality. We treat the following finite configuration.

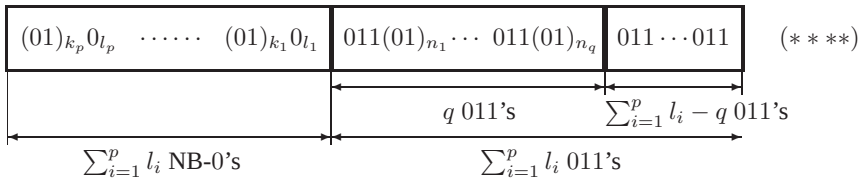


Fig. 7: A finite configuration with $(011)_{\sum_{i=1}^{p} l_i - q}$ added to the right side of the finite configuration given in Fig.6.

For the case of this finite configuration, the moving distance after the p-th stage is

$$m \equiv \sum_{i=1}^{l_1+\cdots+l_p} (n_i + 1) - \sum_{i=1}^{p} k_i = \sum_{i=1}^{q} n_i + \sum_{i=1}^{p} l_i - \sum_{i=1}^{p} k_i,$$

since $n_i = 0$ for $q + 1 \leq i \leq l_1 + \cdots + l_p$.

We add m 01's to the left side of $(****)$ or $-m$ 01's to the right side of $(****)$ according to $m \geq 0$ or $m < 0$, respectively. Here we consider the former case, then we have the following finite configurations.

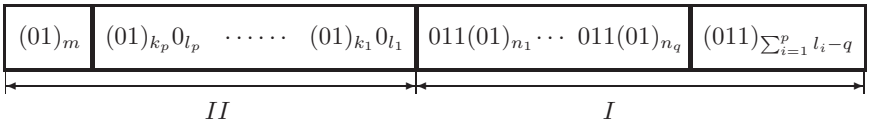


Fig. 8: A finite configuration given by adding $(01)_m$ to the left-hand side of the finite configuration $(***)$.

We may construct a configuration of $\mathcal{Y}$ with placing the above finite configurations I and II as

$$\boldsymbol{a} \equiv \cdots, II, II, \cdots, II, I, \cdots, I, I, \cdots$$

$\boldsymbol{a}$ is periodic, since the middle II and I simultaneously disappear in the time development of $\boldsymbol{a}$ and the trajectory of $\boldsymbol{a}$ returns to $\boldsymbol{a}$. Noticing the parameters p and q used for constructing $\boldsymbol{a}$ is taken arbitrarily from $\boldsymbol{x}$, it is shown that there exists a periodic configuration in an arbitral neighborhood of $\boldsymbol{x}$.

Construction of a transitive configuration is easy after the proof of the dense property of periodic configurations, and then is omitted here.

4 Concluding remarks

In this paper we have examined the dynamical properties of $(\mathcal{A}, \boldsymbol{g}_{56})$ and showed that

(i) $\left(\mathcal{S}_{0(1),1(1,2)}, \boldsymbol{g}_{56}\right)$ and $(\mathcal{S}_{1(1)}, \boldsymbol{g}_{56})$ are left and right subshifts of finite type, respectively, and both of them are topologically mixing and Devaney chaos,

(ii) Interactions between NB-0 and 01, between NB-0 and 011, and between 01 and 011 provide Devaney chaos property of $(\mathcal{Y}, \boldsymbol{g}_{56})$ which is neither left nor right shift.

(iii) $(\mathcal{A}, \boldsymbol{g}_{56})$ has every periodic configurations except period 4.

(iv) The dynamics of $(\mathcal{A}, \boldsymbol{g}_{56})$ is richer than that of $(\mathcal{A}, \boldsymbol{g}_{40})$ because of only one difference in the local transition function corresponding 1 or 0 to 100, respectively.

Full examination of the dynamical properties of rule 56 on $\mathcal{S}_{1(1,2)} \setminus \left(\mathcal{Y} \cup \mathcal{S}_{0(1),1(1,2)} \cup \mathcal{S}_{1(1)}\right)$ remains to be an open problem, but we conjecture $\lim_{t\to\infty} d\left(\boldsymbol{g}_{56}^t(\boldsymbol{x}),\ \mathcal{S}_{0(1),1(1,2)} \cup \mathcal{S}_{1(1)}\right) = 0$ for almost all element $\boldsymbol{x}$ of the set, that to say, the trajectory of $\boldsymbol{x}$ is attracted to $\mathcal{S}_{0(1),1(1,2)} \cup \mathcal{S}_{1(1)}$.

The dynamical properties observed in this paper have not been observed by computer simulation for randomly given initial configurations. We could imagine that there remains a lot of interesting properties of even ECA which are hardly recognized by computer simulation.

References

[1] G. Cattaneo, E.Formenti and L.Margara, Topological Definitions of Deterministic Chaos, In Cellular Automata, eds. M Delorme and J. Mazoyer, Kluwer Academic Publishers(1999), 213-259.

[2] B.P.Kitchens, Symbolic Dynamics: One-sided, Two-sided and Countable State Markov Shifts, Springer(1991).

[3] F. Ohi, Chaotic Properties of the Elementary Cellular Automaton Rule 40 in Wolfram's Class I, Complex Systems, 17 (2007), 295–308.

[4] S. Wolfram, Statistical mechanics of cellular automata, Review of Modern Physics **55** (1983) 601–644.

[5] S. Wolfram, Universality and complexity in cellular automata, Physica **10D** (1984) 1–35.

[6] S. Wolfram, A New Kind of Science, Wolfram Media, Inc. (2002).

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 299–308

Gliders and Ether in Rule 54

Markus Redeker

International Centre of Unconventional Computing, University of the West of England, Bristol, United Kingdom.
Email: markus2.redeker@live.uwe.ac.uk

This is a study of the one-dimensional elementary cellular automaton rule 54 in the new formalism of "flexible time". We derive algebraic expressions for groups of several cells and their evolution in time. With them we can describe the behaviour of simple periodic patterns like the ether and gliders in an efficient way. We use that to look into their behaviour in detail and find general formulas that characterise them.

Keywords: Rule 54, one-dimensional cellular automata, gliders, ether, flexible time

1 Introduction

This is a case study of one specific cellular automaton, Rule 54, with the methods developed in [3]. They were developed to allow the study of cellular automata with the methods of theoretical mathematics and without the need for computer simulations. While the previous paper concentrates on the development of the theory, here the ideas are presented in a less formal way and used to work with larger structures.

Section 2 of this paper introduces the formalism in a less formal way than in [3] and shows how the transition function of the cellular automaton can be expressed in it. The resulting formulas still describe only the behaviour of a small number of cells at a time. Therefore in Section 3 rules for larger groups

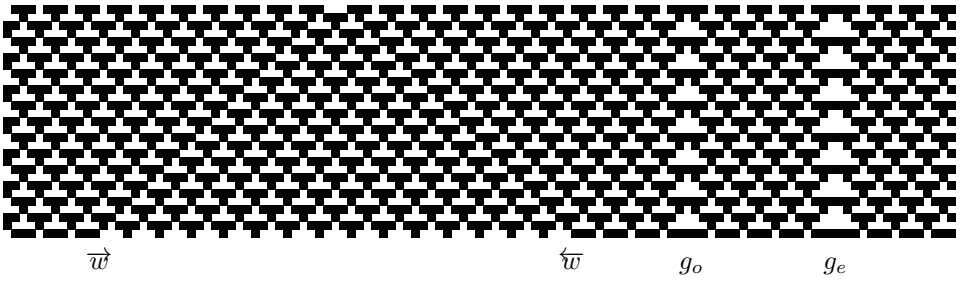


Figure 1: Periodic patterns under Rule 54. The diagram shows three types of gliders on an ether background. Time goes upward. (Since Rule 54 is symmetric, $\overleftarrow{w}$ and $\overrightarrow{w}$ are viewed as variants of the same particle.)

of cells are found. We use them in Section 4 to study the behaviour of four simple periodic structures that occur under Rule 54: the ether and three types of gliders (Figure 1). We find formulas for them and general expressions for gliders and ethers and look into their behaviour.

2 Local Interactions

2.1 Rule 54

"Rule 54" is the common name – following the naming convention of Stephen Wolfram [4] – of a one-dimensional cellular automaton with two states and a three-cell neighbourhood.

At every time it consists of an infinite line of cells. The *state* of each cell is an element of the set $\Sigma = \{0, 1\}$, and the behaviour of the automaton is given by its *local transitions function*

$$\varphi : \Sigma^3 \longrightarrow \Sigma\,. \tag{1}$$

It is applied to every three-cell subsequence of the infinite cell line, and the resulting value is the state of the cell in the middle at the next time step. Rule 54 has

$$\varphi(s) = \begin{cases} 1 & \text{for } s \in \{001, 100, 010, 101\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Sequences of elements of Σ – like 001 – stand here and later for elements of $\Sigma^* = \bigcup_{k\geq 0} \Sigma^k$. Note that φ is symmetric under the interchange of left and right.

2.2 Situations

The formalism of *Flexible Time* [3] is motivated by the idea that it is easier to find patterns in the evolution of cellular automata if one considers structures that involve cells at different times.

These structures are here calles *situations*. They are a generalisation of the sequences of cell states (like 001) considered before. These sequences give the states of neighbouring cells at a certain unspecified time. Thus the sequence 001 describes the states of three cells, possibly at the positions $x = 0$, 1, 2, and tells us that the cells at $x = 0$ and $x = 1$ are in state 0 and the cell at $x = 2$ is in state 1. The position information is implicit in the ordering of the symbols: When a symbol in the sequence stands for the state of a certain cell, its right neighbour in the sequence gives the state of its right neighbour cell, and so on.

Situations are then cell sequences that extend over space and time. To write them down we need additional symbols for a change of time. The symbols we actually use stand for a displacement in time and also in space, because they harmonise then better with the way a cellular automaton evolves.

Under Rule 54, situations are written as sequences of the symbols 0, 1, $\ominus$ and $\oplus$. The intended interpretation can most easily be described in terms of instructions to write symbols on a grid. The fields of the grid are labelled by pairs $(t, x) \in \mathbb{Z}^2$; x is the position of a cell and t a time in its evolution. The writing rules are then

- At the beginning the writing position is at $(0, 0)$.
- If the next symbol is 0 or 1, write it down and move the writing positions one step forward; if it was (t, x) it is now $(t, x + 1)$.
- If the next symbol is $\ominus$, move the writing position from (t, x) to $(t - 1, x - 1)$.
- If the next symbol is $\oplus$, move the writing position from (t, x) to $(t + 1, x - 1)$.

- *No overwriting*: One cannot write different symbols at the same field. (This concerns expressions like $01{\oplus}{\ominus}1$: After writing $01{\oplus}{\ominus}$ one is again at position $(0,1)$ and tries to write a 1 in a field that contains already a 0. So $01{\oplus}{\ominus}1$ is not a valid situation, but $01{\oplus}{\ominus}0$ is.)

The result, in mathematical terms, is a function from a subset of $\mathbb{Z}^2$ to Σ together with an element of $\mathbb{Z}^2$ (the final writing position). The function, which is called π_s for a situation s, describes the states of some cells at some times, while the element of $\mathbb{Z}^2$, written $\delta(s)$, will be important when parts of situations are substituted for others. The whole situation is then the pair $s = (\pi_s, \delta(s))$. We will also need an empty situation, which is written λ.

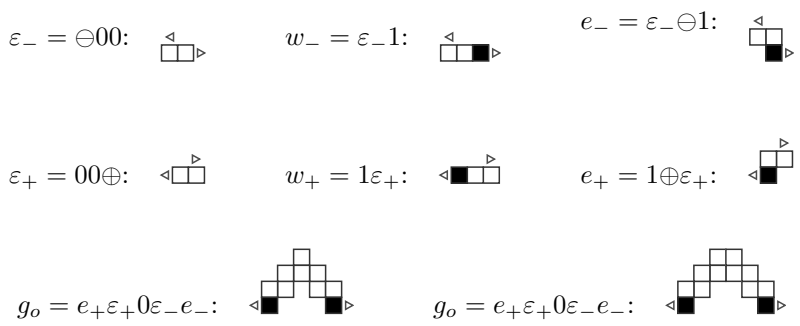


Figure 2: Useful situations in Rule 54.

In Figure 2 you can see diagrams for some situations that will become useful later. Cells in the states 0 and 1 appear as □ and ■, while the initial and final writing position are marked by small triangles: ◁ stands left of the start position, ▷ at the end position. The diagram for g_e becomes less surprising if one notices that $\varepsilon_-\varepsilon_+ = 00{\oplus}{\ominus}00$ has the diagram ◁□□▷: a first case of overwriting.

I have also treated there the situations as normal algebraic expressions, like elements of a semigroup. Product and exponentiation are defined in the usual way: ε^2 is the result of writing ε twice, and so on. However, due to the restrictions on overwriting, not all products of situations exist.

2.3 Reactions

The evolution of cellular automata is described by *reactions*, expressions of the form $a \longrightarrow b$ with two situations a and b. The situation b represents a "partially later" state of the cellular automaton than a, with the states of some cells at a later time than a.

To make this notion more precise, let us consider functions of the form $\pi : E \longrightarrow \Sigma$. They are called *cellular processes* in [3]. If a cellular process fulfills the condition

$$\begin{aligned} \text{If} \quad (t,x-1),(t,x),(t,x+1) \in E \quad &\text{then} \quad (t+1,x) \in E \\ &\text{and} \quad \pi(t+1,x) = \varphi(\pi(t,x-1)\pi(t,x)\pi(t,x+1)), \end{aligned} \tag{3}$$

then it describes a part of the evolution of a cellular automaton under the rule φ.

$\ominus 000 \to 0 \ominus 00$	$000 \oplus \to 00 \oplus 0$
$\ominus 001 \to 1 \ominus 01$	$100 \oplus \to 10 \oplus 1$
$\ominus 010 \to 11 \ominus 0$	$010 \oplus \to 0 \oplus 11$
$\ominus 011 \to 00 \ominus 1$	$110 \oplus \to 1 \oplus 00$
$\ominus 10 \to 1 \ominus 0$	$01 \oplus \to 0 \oplus 1$
$\ominus 11 \to 0 \ominus 1$	$11 \oplus \to 1 \oplus 0$
$00 = 00 \oplus \ominus 00$	$\ominus 00 \oplus \to \lambda$
$01 = 01 \oplus \ominus 01$	$0 \ominus 1 \oplus 0 \to 0$
$10 = 10 \oplus \ominus 10$	$1 \ominus 1 \oplus 1 \to 1$
$11 = 11 \oplus \ominus 11$	

Table 1: Generator reactions for Rule 54

With this notion we can define "$\to$" as a binary relation on the set of situations: $a \to b$ is true if $\delta(a) = \delta(b)$ and for all cellular processes π that fulfill (3) we have: If $\pi \supseteq \pi_a$ then $\pi \supseteq \pi_b$.

One can see that if xay and xby are situations and there is a reaction $a \to b$, then $xay \to xby$ is a reaction too. This is called the *application* of $a \to b$ on xay. We can use that and describe the behaviour of a cellular automaton by a small set of *generator reactions* between a small number of cells. All the others follow from them by application on larger situations and by chaining the reactions. Table 1 shows a set of generator reactions for Rule 54. It is derived from (2) but contains some shortcuts.

To derive Table 1 we start with the rule that

$$\varphi(\alpha\beta\gamma) \to \sigma \qquad \text{becomes} \qquad \ominus\alpha\beta\gamma \to \sigma\ominus\beta\gamma \qquad \text{and} \qquad \alpha\beta\gamma\oplus \to \alpha\beta\oplus\sigma, \tag{4}$$

because then σ is placed correctly one time step later than β. The first two lines of Table 1 are found this way. Other reactions, like $\ominus 10 \to 1 \ominus 0$, are the result of a unification: There would be both $\ominus 100 \to 1 \ominus 00$ and $\ominus 101 \to 1 \ominus 01$, but the state of the rightmost cell has no influence on the result and is therefore left out at both sides of the reaction. These new, shorter reactions can now be applied on the results of some others: $\ominus 010 \to 1 \ominus 10$, a reaction that one gets from (4), is then extended by $1 \ominus 10 \to 11 \ominus 0$ to $\ominus 010 \to 11 \ominus 0$. With these methods the top block of Table 1 is derived.

The purpose of the equations and reactions at the bottom of Table 1 is to create and destroy $\ominus$ and $\oplus$ symbols. The destruction reactions at the right remove also cell states that cannot be used in another reaction.[(i)]

Together the reactions of Table 1 define a *reaction system* Φ. It consists of a set of situations and the reactions between them. We use a common convention and write $s \in \Phi$ if s is an element of the set of reactions of Φ.

3 A Reaction System with Triangles

3.1 Triangles

Now we need rules for larger structures. If their behaviour is understood, we can find reaction that simulate them in one step. At the present stage these structures will be periodic sequences of cells, and we start with the simplest of then, the sequences in which all cells are in the same state.

[(i)] In [3], which uses a slightly other definition of situations, the equations would have to be written as reactions. The destruction

(a) Triangle (b) Reaction $1^7 \to 1\oplus(0^2\oplus)^3 0(\ominus 0^2)^3\ominus 1$

Figure 3: Triangles and triangle reactions

There are only two of them, and we evolve them first for only one time step.

$$0^k \to 0^2\oplus 0^{k-2}\ominus 0^2 \qquad k \geq 0 \tag{5}$$
$$1^k \to 1\oplus 0^k\ominus 1 \qquad k \geq 0 \tag{6}$$

We can see that 0^k is a *persistent* pattern that reappears in the next time step, while 1^k is *instantaneous* and exists only for one time step. Our guiding principle for a new, faster reaction system will be that evolution should never stop when a persistent pattern is reached.

So both reactions should be continued. The result for (5) – and therefore also for (6) – depends on the parity of k and is best expressed as

$$0^{2k+\iota} \to (0^2\oplus)^k 0^\iota (\ominus 0^2)^k \qquad k \geq 1, \iota \in \{0,1\}, \tag{7}$$
$$1^{2k+\iota} \to 1\oplus(0^2\oplus)^k 0^\iota (\ominus 0^2)^k\ominus 1 \qquad k \geq 1, \iota \in \{0,1\}\,. \tag{8}$$

They are both examples of *triangle reactions*, that are reactions of the general form

$$a_- x^k b_+ \to a_+ y_+^k c y_-^k b_- \qquad k \geq 0, \tag{9}$$

which trace the boundaries of a space-time triangle.[(ii)] Figure 3 shows an example.

Since the "boundary terms" of the triangles will occur often, we will use abbreviations for them,

$$\varepsilon_- = \ominus 0^2, \qquad \varepsilon_+ = 0^2\oplus, \qquad e_- = \ominus 0^2\ominus 1, \qquad e_+ = 1\oplus 0^2\oplus\,. \tag{10}$$

The definitions for e_- and e_+ have been chosen with the benefit of hindsight – instead of choosing abbreviations for $\ominus 1$ and $1\oplus$ – because these terms will be important later. With them (7) and (8) become

$$0^{2k+\iota} \to \varepsilon_+^k 0^\iota \varepsilon_-^k \qquad k \geq 1, \iota \in \{0,1\}, \tag{11}$$
$$1^{2k+\iota} \to e_+ \varepsilon_+^{k-1} 0^\iota \varepsilon_-^{k-1} e_- \qquad k \geq 1, \iota \in \{0,1\}\,. \tag{12}$$

3.2 Destruction of Boundary Terms

We must now extend these reactions to a full reaction system. Since (11) and (12) create the boundary terms ε_-, ε_+, e_- and e_+, the new reactions should destroy them. To keep the number of new reactions small, we require that the triangle reactions are always used efficiently and never applied to only a part of a cell sequence. (A reaction like $0^3 \to \varepsilon_+\varepsilon_- 0$ will be forbidden then.) We may express that by the

reactions, which are chosen somewhat ad hoc, are also different from the result of the result of the rules given there.

[(ii)] We can bring reaction (12) in that form by setting $a_- = 1^{2+\iota}$, $x = 1^2$, $a_+ = \lambda$, $y_+ = 0^2\ominus$ $y_+ = \ominus 0^2$ and $c = 0^\iota$.

States:	$0, 1, \varepsilon_-, \varepsilon_+, e_-, e_+$.	
Situations:	No subsequences $\varepsilon_- 0$, $0\varepsilon_+$, $e_- 1$, $1e_+$.	
Triangles:	$0^{2k+\iota} \to \varepsilon_+^k 0^\iota \varepsilon_-^k$,	$k \geq 1, \iota \in \{0, 1\}$
	$1^{2k+\iota} \to e_+ \varepsilon_+^{k-1} 0^\iota \varepsilon_-^{k-1} e_-$,	$k \geq 1, \iota \in \{0, 1\}$
Boundary terms:	$\varepsilon_-(10)^k 1\varepsilon_+ \to 1^{2k+3}$	$k \geq 0$
	$\varepsilon_-(10)^k e_+ \to 1^{2k+1}\varepsilon_+$	$k \geq 0$
	$e_-(01)^k \varepsilon_+ \to \varepsilon_- 1^{2k+1}$	$k \geq 0$
	$e_-(01)^k 0e_+ \to \varepsilon_- 1^{2k+1}\varepsilon_+$	$k \geq 0$
	$\varepsilon_-\varepsilon_+ \to \varepsilon_+\varepsilon_-$	
	$e_- e_+ \to \varepsilon_+\varepsilon_-$	

Table 2: Rule 54 in triangle form

requirement that the situations may never contain the terms $\varepsilon_- 0$, $0\varepsilon_+$, $e_- 1$ or $1e_+$: they would be the result of such an incomplete application.

It will be enough for a working system to consider reactions that start from terms of the form $b_- s b_+$, with $b_- \in \{\varepsilon_-, e_-\}$, $b_+ \in \{\varepsilon_+, e_+\}$, $s \in \Sigma^*$, to which no other reactions are applicable. The last requirement means that s must consist of cells in states 0 and 1 in alternating order: Two cells in the same state are already the starting point of a triangle reaction. It turns out that there are only six types of reactions that satisfy this requirement and that of the forbidden subconfigurations in the previous paragraph.

Here they are, together with reactions that start from them:

$$\varepsilon_-(10)^k 1\varepsilon_+ \to 1^{2k+3} \qquad k \geq 0, \tag{13}$$

$$\varepsilon_-(10)^k e_+ \to 1^{2k+1}\varepsilon_+ \qquad k \geq 0, \tag{14}$$

$$e_-(01)^k \varepsilon_+ \to \varepsilon_- 1^{2k+1} \qquad k \geq 0, \tag{15}$$

$$e_-(01)^k 0e_+ \to \varepsilon_- 1^{2k+1}\varepsilon_+ \qquad k \geq 0, \tag{16}$$

$$\varepsilon_-\varepsilon_+ \to \varepsilon_+\varepsilon_-, \tag{17}$$

$$e_- e_+ \to \varepsilon_+\varepsilon_- . \tag{18}$$

The first four reactions have been chosen minimally such that the cell states of s in $b_- s b_+$ are replaced with states that are exactly one time step later, such as in (5) and (6). The last two reactions cover the situations with $s = \lambda$ that are not special cases of the previous four reactions. The resulting reactions system is listed in Table 2.

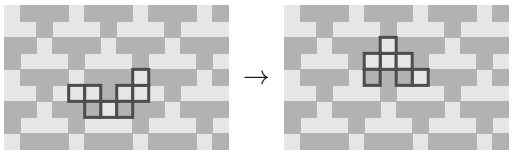
4 Ether and Gliders

4.1 The Ether

Now we will use the new reaction system to look at some phenomena that occur under Rule 54. The first of them is the *ether*, a robust background pattern. It consists at alternating time steps of either the cell sequence 01^3 or 10^3 infinitely repeated. (To verify the reactions in this section Table 3 may be helpful.)

$\varepsilon_- e_+ \to 1 e_+$	$\varepsilon_- e_+ \to w_+$
$e_- \varepsilon_+ \to e_- 1$	$e_- \varepsilon_+ \to w_-$
$\varepsilon_- 1 \varepsilon_+ \to e_+ 0 e_-$	$w_- e_+ \to e_+ 0 e_-$
	$e_- w_+ \to e_+ 0 e_-$
$e_- 0 e_+ \to e + 0 e_-$	
$\varepsilon_- 10 e_+ \to 1^3$	$w_- 0 e_+ \to 1^3$
$e_- 10 \varepsilon_+ \to 1^3$	$e_- 0 w_+ \to 1^3$
$\varepsilon_- 101 \varepsilon_+ \to 1^5$	$w_- w_+ \to 1^5$

Table 3: Simple Reactions that are useful in Section 4. Most of them are special cases of Table 2 or derived from them.



$$e_- 0 e_+ 0 \to e_+ 0 e_- 0$$

Figure 4: How the ether reaction fits into the development if the ether. The cells that belong to the reaction are marked.

In the reaction system a formula for the ether can be derived from the 01^3 generation: We have

$$01^3 \to 0 e_+ 0 e_- \tag{19}$$

and (see Figure 4)

$$0 e_- 0 e_+ \to 0 e_+ 0 e_-, \tag{20}$$

therefore

$$(01^3)^k \to (0 e_+)^k (0 e_-)^k \qquad k \geq 0, \tag{21}$$

a very simple triangle reaction. This is in contrast to the other possible starting point, 10^3, where one gets

$$(10^3)^k \to 1 \varepsilon_+ (0 e_+)^k (0 e_-)^k \varepsilon_- 1 \qquad k \geq 1, \tag{22}$$

a more complicated triangle reaction, in which also the components of the other ether phase, e_- and e_+, reappear. The reaction system selects thus one of the phases of the ether as more natural than the other, which is a helpful simplification.

If one now looks back at (17) and compares it with (20), one sees that they follow a common pattern. Both are *background reactions* of the form

$$b_- b_+ \to b_+ b_- \,. \tag{23}$$

This reaction can easily be iterated to $b_-^k b_+^\ell \to b_+^\ell b_-^k$, which describes the evolution of a large piece of a periodic background pattern.

Their involvement in the ether is the reason why e_- and e_+ got their names in (10).

4.2 Gliders

There are three kinds of long-lived structures that are described in [1] in some detail. There they are called particles, now usually *gliders*. There is one moving particle w, which appears as $\overleftarrow{w}$ and $\overrightarrow{w}$, depending on the direction in which it moves, and the "odd" and "even gutter" g_0 and g_e, which are immobile.

The w particle "may be generated by three 0's followed by three 1's or the converse" [1, p. 870]. We try this now and get

$$0^3 1^3 \to \varepsilon_+ 0 \varepsilon_- e_+ 0 e_- \to \varepsilon_+ 0\ 1 \varepsilon_+\ 0 e_-. \tag{24}$$

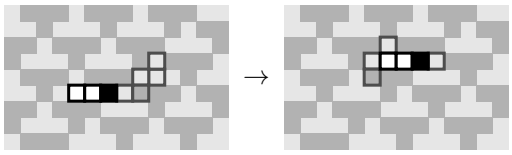
In it we can recognise $0e_-$ as a part of the ether and $\varepsilon_+ 0$ as a part of the ether in the wrong phase (as in (20) and (22)), so the rest must be the w particle. Therefore we define

$$w_- = \varepsilon_- 1, \qquad w_+ = 1 \varepsilon_+. \tag{25}$$

These definitions must be verified: We must show that w actually moves through the ether. But we have

$$w_- 0 e_+ 0 = \varepsilon_- 1 0 e_+ 0 \to 1^3 \varepsilon_+ 0 \to e_+ 0 e_- \varepsilon_+ 0 \to e_+ 0 \varepsilon_- 1 0 = e_+ 0 w_- 0, \tag{26}$$

which shows how w_- is destroyed and reappears at the right of its previous position (Figure 5). w_- is therefore stable and corresponds to the right-moving glider $\overrightarrow{w}$ of [1].



$$w_- 0 e_+ 0 \to e_+ 0 w_- 0$$

Figure 5: A $\overrightarrow{w}$ glider moving on an ether background. The w_- part is emphasised.

The two immobile gliders, g_e and g_o, are in fact small triangles, as can be seen from the pictures in [1]. It turns out that the right definitions for them are

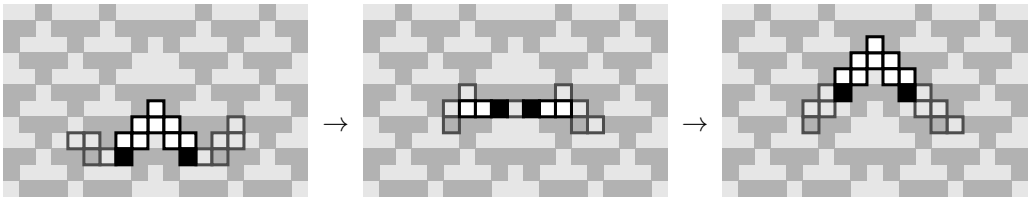
$$g_o = e_+ \varepsilon_+ 0 \varepsilon_- e_-, \qquad g_e = e_+ \varepsilon_+^2 \varepsilon_-^2 e_-. \tag{27}$$

The verification that they actually behave like gliders is straightforward (Figure 6),

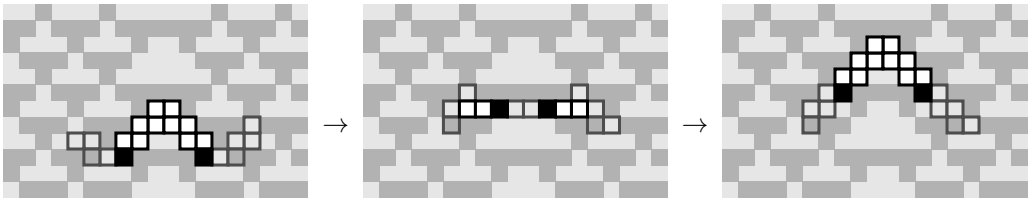
$$\begin{aligned} e_- 0 g_o 0 e_+ 0 &= e_- 0 e_+ \varepsilon_+ 0 \varepsilon_- e_- 0 e_+ 0 \\ &\to e_+ 0\ e_- \varepsilon_+ 0 \varepsilon_- e_+ 0\ e_- 0 \\ &\to e_+ 0\ w_- 0 w_+ 0\ e_- 0 \\ &\to e_+ 0\ 1^5 0\ e_- 0 \\ &\to e_+ 0\ e_+ \varepsilon_+ 0 \varepsilon_- e_- 0\ e_- 0 = e_+ 0 g_o 0 e_- 0, \end{aligned} \tag{28}$$

$$\begin{aligned} e_- 0 g_e 0 e_+ 0 &= e_- 0 e_+ \varepsilon_+^2 \varepsilon_-^2 e_- 0 e_+ 0 \\ &\to e_+ 0\ e_- \varepsilon_+^2 \varepsilon_-^2 e_+ 0\ e_- 0 \\ &\to e_+ 0\ w_- \varepsilon_+ \varepsilon_- w_+ 0\ e_- 0 \\ &\to e_+ 0\ 1^6 0\ e_- 0 \\ &\to e_+ 0\ e_+ \varepsilon_+^2 \varepsilon_-^2 e_- 0\ e_- 0 = e_+ 0\ g_e 0\ e_- 0, \end{aligned} \tag{29}$$

but the appearance of the w gliders in the process is a bit surprising. It suggests the interpretation that the gliders g_o and g_e decay into two w gliders, which then collide and create its next incarnation. With flexible time the gliders suddenly have an internal structure.



$$e_-0g_o0e_+0 \to e_+0w_-0w_+0e_-0 \to e_+0g_o0e_-0.$$



$$e_-0g_e0e_+0 \to e_+0w_-\varepsilon_-\varepsilon_+w_+0e_-0 \to e_+0g_e0e_-0.$$

Figure 6: Evolution of the g_o and g_e gliders, together with the intermediate states where the w gliders appear.

The three glider reactions (26), (28) and (29) have again a common structure, which can be described by the *glider condition*

$$b_-^k g b_+^\ell \to b_+^\ell g b_-^k. \tag{30}$$

Here b_- and b_+ form a background pattern as in (23) and g is the glider. The number $(\ell - k)/(\ell + k)$ is a measure for the speed of the glider.

We have now already touched the creation of other gliders by the w gliders. Of the two syntheses found in the behaviour of the g particles, the first one,

$$w_-0w_+ \to g_o, \tag{31}$$

is more important because here the w gliders are at the right distance to have been part of the ether before. Such a glider synthesis has been already noticed in [1], but here it occurs as a corollary of a previous analysis.

References

[1] N. Boccara, J. Nasser, M. Roger. *Particlelike structures and their interactions in spatiotemporal patterns generated by one-dimensional deterministic cellular-automaton rules.* Physical Review A 44 (1991), 866–875.

[2] Genaro Juárez Martínez, Andrew Adamatzky, Harold V. McIntosh. *Phenomenology of glider collisions in cellular automaton rule 54 and associated logical gates.* Chaos, Fractals and Solitons 28, 100–111 (2006).

[3] Markus Redeker, *Flexible Time and the Evolution of One-Dimensional Cellular Automata.* Journal of Cellular Automata (to appear), `http://arxiv.org/abs/0812.4242`.

[4] Stephen Wolfram, *Universality and Complexity in Cellular Automata.* Physica 10D (1984), 1–35.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 309–318

Dynamics of 1-d cellular automa with distance-dependent delays

Thimo Rohlf[1,2] and Jürgen Jost[2]

[1] *Epigenomics Project, ISSB, Genopole Campus 1, Genavenir 6, 5 Rue Henri Desbrueres, F-91034 Evry cedex, France*
[2] *Max-Planck-Insititute for Mathematics in the Sciences, Inselstr. 22, D-04103 Leipzig, Germany*

Delays in signal transmission are found in many complex systems in nature, e.g. as a consequence of spatial distance between the elements the system consists of. In standard cellular automata (CAs), however, usually an instantaneous transmission of information in the update-neighborhood of cells is assumed, and distance information is disregarded. The objective of this study is to overcome this limitation by a generalization of the CA update scheme. We investigate the effect of spatio-temporal delay depending linearly on the distance between cells in synchronously updated, one-dimensional CAs. We find that delays induce distinctive transitions between different classes of dynamical behavior, and on average tend to increase the space-time entropy of CA patterns. A more detailed investigation also taking into account mutual information shows that transitions in the opposite direction are also present in considerable proportions, indicating a rich space of rule-dependent dynamical transitions induced by delays.

Keywords: 1-d cellular automata, delays, entropy, mutual information

1 Introduction

Complex spatio-temporal patterns are often found in nature, yet in many cases their emergence can be explained by surprisingly simple dynamical systems of locally interacting elements. Systems of this kind are found, e.g., in physical chemistry (e.g. reaction-diffusion systems), in biology (e.g. in morphogenesis of multicellular organisms) and even in sociology and economy. Traditionally, many of these systems are modeled by sets of coupled partial differential equations, which allow a detailed investigation of possible dynamical behaviors. However, these approaches often involve a large number of free parameters, and reach the limits of computational tractability quickly, in particular, if systems involve complex dependencies (networks) between dynamical elements. Furthermore, in many cases dynamical transitions can be well approximated by discrete state spaces, which sometimes drastically reduces the complexity of the problem; examples are found in physics, e.g. the Ising model of magnetism, in biology, e.g. the relatively small set of distinct cell types and the well defined transitions between them (differentiation), and in economy, e.g. the complex dynamics of stock exchanges generated from basically binary ”buy” or ”sell” decisions. Very often, the geometric space in which such discrete interactions take place has a strong impact on dynamics; this is explicitly taken into account in *Cellular Automata*. Besides highly

regular, grid-based topologies, that have been extensively explored, recently also CA topologies with partially randomized structure ("small world" CAs) have come into the focus of research (Marr and Hütt, 2005).

Originally introduced by *von Neumann* as a theoretical framework for self-replication (von Neumann, 1966), it was realized that CAs could serve as models for much broader classes of phenomena. It was shown, for example, that Conway's famous "game of life" (Berlekamp et al., 1982)) as well as simple one-dimensional cellular automata (Lindgren and Nordahl, 1990) are capable of universal computation in the sense of a Turing machine. Based on methods developed in statistical mechanics, detailed studies on the simplest class of CA, elementary 1D CA with $k = 2$ states and 3-cell neighborhood, were carried out in the 1980's. Wolfram (Wolfram, 1983, 1984) developed a qualitative classification scheme of the $2^{2^3} = 256$ elementary CA rules that distinguished four different 'complexity classes' of their dynamics (class I: fixed-point attractors, class II: space-time periodic attractors (limit cycles), class III: aperiodic space-time chaos, class IV: 'complex' dynamics, i.e. traveling, localized aperiodic structures on regular background). A number of methods was developed to obtain a more quantitative characterization of CA dynamics, e.g. mean field models (Schulman and Seiden, 1978), local structure theory (Gutowitz et al., 1987), and quantification of pre-images (Soto, 2008). Recently, the investigation of different classes of CA update schemes has come into the focus of research, showing, for example, that stochastic, asynchronous updates can induce distinctive phase transitions in standard CAs that can be well described with methods from Statistical Physics (Fatès, 2009).

While on one hand the drastic simplification provided by CA models can have obvious benefits, on the other hand their very idealized dynamics also limits their range of applicability. In real spatially extended systems, for example, a delayed coupling of dynamical elements naturally emerges from their spatial distance. It has been shown that delays can substantially alter the phase space of dynamical systems (Atay et al., 2004; Atay and Karabacak, 2006). In this paper, we generalize the class of standard one-dimensional CAs by introduction of delays in signal transmission that depend linearly on the distance between cells, defining a new class of dynamical systems that we call delay cellular automata (DCAs). A comparison with CAs where delays are not present indicates that distinctive dynamical transitions are induced by delays, and on average tend to increase the space-time entropy of CA patterns. A more detailed investigation also taking into account mutual information shows that transitions in the opposite direction are also present in considerable proportions, indicating a rich space of rule-dependent dynamical transitions induced by delays.

2 Model and Definitions

2.1 One dimensional CA without delay

Let us first start with cellular automata without delay. Consider a one-dimensional cellular automaton (CA) with parallel update. N cells are arranged on a one-dimensional lattice, and each cell is labeled uniquely with an index $i \in \{0, 1, ..., N-1\}$. Each cell can take k possible states $\sigma_i \in \Sigma := \{0, 1, .., k-1\}$. CA dynamics is defined by a map

$$f : \{0, 1, ..., k-1\}^n \mapsto \{0, 1, ..., k-1\} \tag{1}$$

that determines the state $\sigma_i(t)$ of cell i as a function of its own state $\sigma_i(t-1)$ and the states of its $n-1$ closest neighbors at time $t-1$; n (odd) is also called the ***neighborhood size*** of the CA and $r := (n-1)/2$

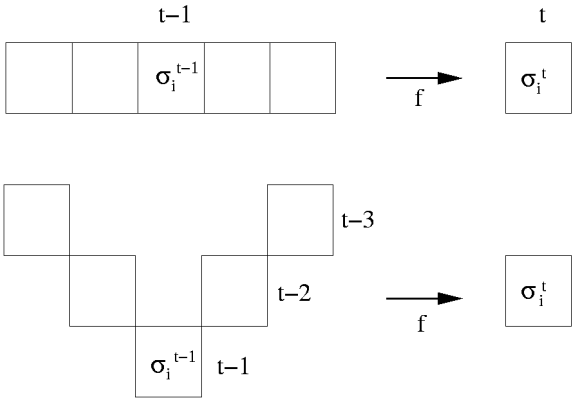


Fig. 1: Schematic comparison of the conventional CA update scheme (upper panel) and of the DCA update scheme (lower panel), for the example of a CA with neighborhood size $n = 5$. Whereas in CA without delays, for each cell i its own state and the states of its four closest neighbors at time $t - 1$ are mapped to a new state at time t, in DCA the cells own state at time $t - 1$, its nearest neighbors' states at time $t - 2$, and its next-nearest neighbors states at time $t - 3$ are considered.

the *radius* of the CA. The map f then reads as

$$\sigma_i(t) = f[\sigma_{i-r}(t-1), ..., \sigma_i(t-1), ..., \sigma_{i+r}(t-1)]. \tag{2}$$

f is also called the rule table of the CA; the N cells are updated in parallel by application of this rule table to each single cell. Notice that we restrict ourselves to *symmetric neighborhoods*, which for many pattern formation problems as observed in nature is the natural choice.

Boundary conditions for the sites $i = 0$ and $i = N - 1$ have to be specified explicitly. In this paper, we always apply periodic boundary conditions.

2.2 One dimensional CAs with spatio-temporal delay

We now introduce *spatio-temporal delays* into CA dynamics (a schematic description is also shown in Fig. 1.) For this purpose, we assume that the time needed for signal transmission increases linearly with the distance between cells. Hence, CA dynamics now is defined by a map

$$g : \{0, 1, ..., k-1\}^n \mapsto \{0, 1, ..., k-1\} \tag{3}$$

which, for the simplest case of delay directly proportional to the distance of the respective neighbor cells, is given by

$$\begin{aligned} \sigma_i(t) \quad = \quad & g[\sigma_{i-r}(t-r-1), \sigma_{i-r+1}(t-r), ..., \sigma_i(t-1), ... \\ & ..., \sigma_{i+r-1}(t-r), \sigma_{i+r}(t-r-1)]. \end{aligned} \tag{4}$$

Notice that the rule tables defined by g do not differ from those defined by f, the dynamics, however, is essentially different due to the delays introduced into the map. Henceforth, we will call this new class of dynamical systems *delay cellular automata (DCA)*.

Further, we mention that, due to the delays that introduce a time dependence on the previous $r+1$ system states, in DCA initialization there does not exist the notion of a uniquely defined initial state, rather, one has to define an *ordered initial set* $\{\Sigma_{ini}\}$ of $r+1$ system states:

$$\{\Sigma_{ini}\} = (\Sigma(t=-r), ..., \Sigma(t=0)) \tag{5}$$

Starting from this initial set, DCA dynamics then is iterated for $t \geq 0$. In our simulations, we mostly set all $r+1$ states to the same (randomly generated) initial state; however, setting $\{\Sigma_{ini}\}$ to $r+1$ *different* states basically leads to the same results.

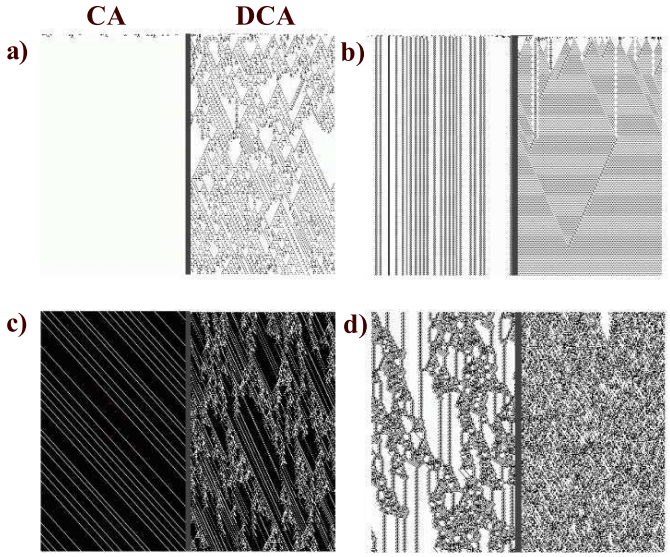
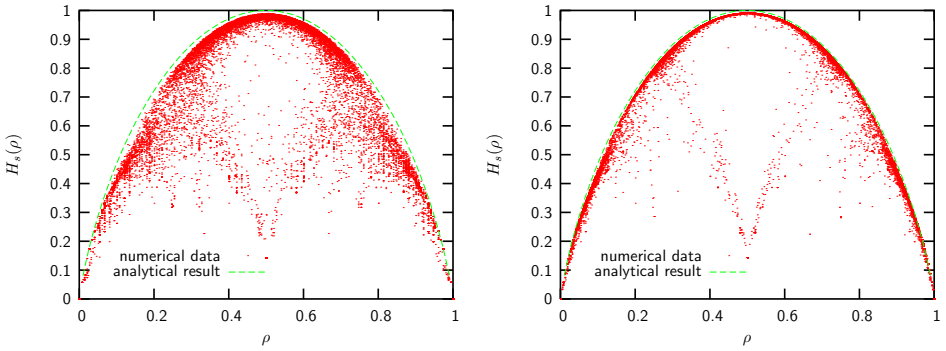


Fig. 2: Space-time diagrams of cellular automata dynamics for four randomly sampled CA rules with $k=2$ and $n=7$; time is running from top to bottom. In each panel, on the left side CA dynamics without delays is shown, whereas on the right side the dynamics for the same rule with delays is shown. The following transitions are frequently observed: fixed point $\rightarrow$ complex, aperiodic dynamics (a), local domains of oscillations $\rightarrow$ (almost) globally synchronized oscillations (b), traveling waves $\rightarrow$ complex aperiodic patterns (c), complex patterns $\rightarrow$ chaotic pattern (d).

3 Comparing the dynamics of DCAs to CAs without delays

Distance-dependent delays, as defined in section 2.2, introduce additional levels of time dependence into CA dynamics, which can also be considered as an implicit memory of size $r+1$ (in addition, a local average over the last $r+1$ states is taken that depends both on time and space). In CAs without delays,



(a) Block entropy of CAs (b) Block entropy of DCAs

Fig. 3: Left panel: The block entropy H_s as a function of the density of 1's, ρ, in CAs without delays, numerical data (red) sampled over 10^7 randomly generated CAs with $k = 2$, $n = 7$ and $N = 150$, compared to H_s^{upper}. Right panel: $H_s(\rho)$ in DCAs, numerical data (red) sampled over 10^7 randomly generated CA with $k = 2$, $n = 7$ and $N = 150$, compared to H_s^{upper}.

in the extreme case each cell 'forgets' its local predecessor states after one dynamical update, in DCA complete loss of memory takes at least $r + 1$ time steps. It has been shown that *explicit* inclusion of memory into CA dynamics can lead to drastic changes of dynamical behavior (Rohlf and Tsallis, 2007), hence, we also expect that spatio-temporal delays substantially change CA dynamics.

Let us now compare the dynamics of DCA with its conventional counterpart, i.e. the corresponding CA rules without delays. Figure 2 shows space-time diagrams of cellular automata randomly sampled from the rule space of CAs with $n = 5$ and $k = 2$. At first visual inspection, the diagrams suggest a general trend that delays tend to increase the complexity of space-time patterns for rules that lead to very simple asymptotic dynamics without delays. For example, delays can induce a transition from a fixed-point pattern to complex triangular patterns (Fig. 2 a)), or improve the synchronization among cells (Fig. 2 b)). On the other hand, "complex" patterns suggesting to belong to class IV in Wolfram's classification scheme tend to become much more randomized when delays are present (Fig. 2 d)). The quantitative and conclusive classification of "complex behavior" in CAs is a persistent problem in CA literature, for which a number of approaches have been suggested (Gutowitz et al., 1987; Li et al., 1990; Sakai et al., 2004). Here, we cannot go into the details of this matter and hence restrict ourselves to a rather coarse-grained classification of DCA in comparison to CAs without delays by application of methods derived from Statistical Mechanics.

3.1 Analysis of block entropies

Any analysis of the complexity of CA dynamics is limited by the "combinatorial explosion" of the number of different possible rule tables even for quite moderate values of n and k. For a k-state CA with neighborhood size n, i.e. k^n different neighborhood configurations, there exist $Z_k^n := k^{k^n}$ different rules tables, leading e.g. to $Z_3^5 = 3^{3^5} \approx 8.71 \cdot 10^{115}$ for a 3-state CA with neighborhood size $n = 5$. Therefore, one needs to identify quantities that can describe the important features of CAs and that can distinguish different types of qualitative behavior of CAs. For instance, the basic information capacity of a CA can

be characterized using the *Shannon entropy H*, which for a discrete process A of k states is given by

$$H(A) = -\sum_{i=0}^{k-1} p(i)\ \log p(i). \tag{6}$$

Here, $p_i \in [0,1]$ is the (asymptotic) probability for a cell j of a given CA to have the state $\sigma_j = i$. This measure, however, disregards all spatial information. Since our new class of dynamical systems operates based on *spatio-temporal delays*, we have to include at least a coarse-grained description of space into our statistical analysis.For this purpose, we consider the spatial *block entropy* of a cellular automaton T Jost (2005):

$$H_s(T) = \lim_{B\to\infty} \lim_{\nu\to\infty} -\frac{1}{B} \sum_{\alpha=1}^{2^B} p_\nu(\alpha) \log p_\nu(\alpha) \tag{7}$$

Here, $p_\nu(\alpha)$ are the relative frequencies with which blocks α of length B of values at consecutive sites appear at time ν. In practice, neither of the two limits can be really taken, and one has to find some compromise. A natural choice for the considered block size e.g. could be $B = n$, i.e. the size of the update neighborhood of each cell, which at the same time yields the "rule entropy" (i.e. the relative frequencies of usage of the different entries in a given rule table). An upper bound of $H_s(T)$ as a function of the average stationary density ρ of 1's in the pattern can be calculated analytically by a mean field theory (Schulman and Seiden, 1978):

$$H_s^{upper}(\rho) = \frac{\rho\ \ln\rho - (1-\rho)\ \ln(1-\rho)}{\ln 2} \tag{8}$$

Figure 3 shows numerically measured values for ensembles of randomly generated CA rules with $k = 2$ and $n = 7$, compared to the result of Eqn. 8. We find that, for most rules, H_s is increased when delays are present, i.e. the block entropy is moved towards the upper bound compatible with the stationary density ρ of 1's in the pattern, indicating increased mixing in phase space. This observation confirms the general impression obtained from Fig. 2.2 with the typical delay-induced transitions fixed point/periodic pattern $\rightarrow$ 'complex' pattern and 'complex pattern' $\rightarrow$ randomized (chaotic) pattern.

3.2 Further classification of dynamical transitions by changes in mutual information

The information about dynamical transitions in CA dynamics induced by delays, quantified by consideration of (block-)entropies in the previous section, is limited in a number of regards. Entropy is a rather coarse measure for information content of dynamics, since it not very sensitive to correlations; furthermore, we so far considered ensembles of DCAs and CAs without dynamics seperately, hence loosing the information of delay-induced effects for a particular update rule. Consequently, we are now interested in a direct comparison of DCA and CA dynamics for given update rules, applying a measure that is more sensitive to spatial and temporal correlations in dynamics, namely *mutual information*. The *spatial mutual information* I_s between two CA cells A and B is defined as

$$I_s(A,B) = H(A,B) - H(A) - H(B), \tag{9}$$

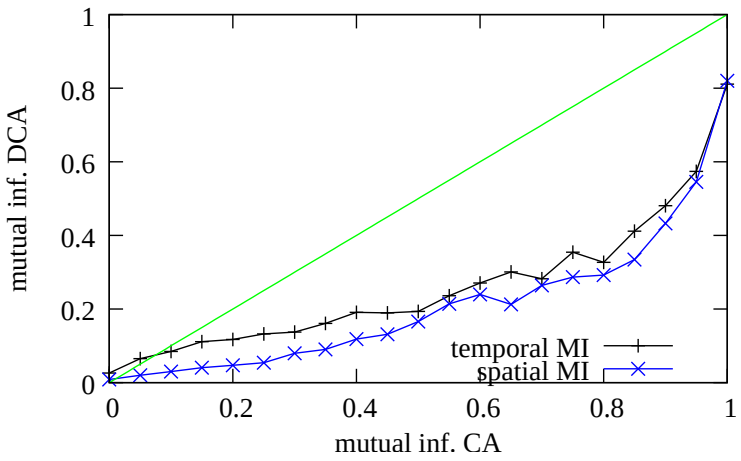


Fig. 4: Average mutual information for dynamical updates with delays, compared to the same set of rules without delays. Statistics was averaged over 80000 rules with $n = 5$, $k = 2$ and $N = 150$, for 1000 updates after a transient of 1000 updates, starting from 250 random initial states for each rule. Bin size for averaging was 0.05.

where $H(A, B)$ is the joint entropy

$$H(A, B) = -\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} p(i, j), \log p(i, j), \tag{10}$$

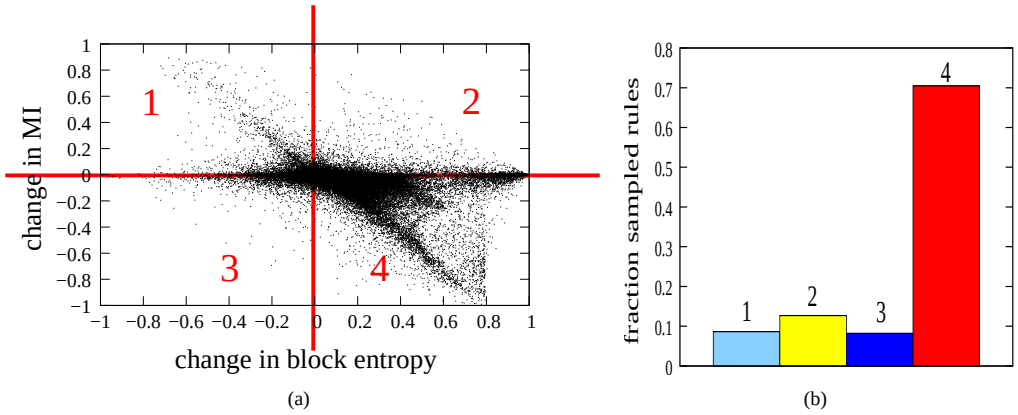
with $p(i, j)$ as the joint probability for co-occurrence of state i and j in cells A and B, respectively. $H(A)$ and $H(B)$ are the single-cell Shannon entropies, as defined in Eq. 6. Similarly, the temporal (future-past) mutual information I_t for a cell A is defined as

$$I_t(A(t), A(t-\tau)) = H(A(t), A(t-\tau)) - H(A(t)) - H(A(t-\tau)), \tag{11}$$

where τ is the number of update steps one looks back in the past.

Figure 4 exhibits the average statistical dependence between the mutual information of 80000 randomly generated CA rules with $n = 5$ and $k = 2$, for updates without delays ($x-$axis) and with delays ($y-$axis). Obviously, the average trend is that delays tend to reduce correlations (and hence mutual information) quite considerably, as becomes evident from the fact that the curve, except for very small values of the temporal mutual information, is always below the line $I_{DCA} = I_{CA}$. However, as became already apparent in section 3, this average picture may be deceptive and may hide the richness of possible dynamical transitions.

A more detailed account of delay-induced changes in dynamical behavior is provided in Fig. 5 (a), which correlates the delay-induced change in block entropy (which can be positive or negative) to the change in temporal mutual information; since a decrease (increase) in entropy indicates more ordered (more chaotic) dynamics, while an increase in MI is indicative of more ”complex” dynamics (i.e. increased correlations), the four quadrants of this diagram can be interpreted with respect to the nature of



(a) (b)

Fig. 5: Left panel: Change in temporal mutual information under DCA update relative to CA update without delays, as a function of the change in block entropy. 80000 randomly sampled rules with $n = 5$, $k = 2$ and $N = 150$ are shown, mutual information ($\tau = 10$) and entropy where averaged over 1000 updates after a transient of 1000 updates, starting from 250 random initial states for each rule. The four quadrants, as indicated, allow a coarse classification of dynamical transitions: 1: chaotic → complex or ordered, 2: ordered → complex, 3: complex or chaotic → ordered, 4: ordered or complex → chaotic. Right panel: frequency distribution of dynamical transitions, numbers referring to the four quadrants of the left panel.

dynamical transitions: 1: chaotic → complex or ordered, 2: ordered → complex, 3: complex or chaotic → ordered, 4: ordered or complex → chaotic. Figure 5 (b) shows a statistics of the distribution of rules into the four quadrants. Evidently, as we already concluded, the transition from ordered or complex to chaotic dynamics is most frequent, however, the opposite transitions are also present in considerable proportions, indicating a rich space of rule-dependent dynamical transitions induced by delays.

4 Discussion

Distance-dependent delays in signal transmission naturally emerge in most spatially extended dynamical systems, however, were neglected in cellular automata models even when neighborhood sizes substantially larger than 3 cells were considered. We introduced distance-dependent *delays* into one-dimensional cellular automata, and thereby defined the new class of *delay cellular automata (DCA)*. Our results indicate that this type of delays considerably changes the dynamics of cellular automata as a consequence of the *implicit local memory* it creates. Very often, this leads to transitions to a different class of asymptotic dynamical behavior, when we compare a given CA rule under iteration without or with delays, for example fixed point → complex aperiodic pattern, periodic pattern → randomized (chaotic) pattern. A general trend is that, for most CA rules, delays increase the space-time entropy of the system, as quantified by measurements of block-entropies of observed space-time patterns. However, a more detailed investigation also taking into account mutual information in space and time, showed that transitions in the opposite direction are also present in considerable proportions, indicating a rich space of rule-dependent dynamical transitions induced by delays.

Our results indicate that, if cellular automata are considered as serious models of spatially extended

dynamical systems, effects that naturally result from spatial extension - here delays - cannot be neglected, since they considerably change the dynamics. Similar observations were made for other classes of dynamical systems, for example, coupled logistic maps (Atay et al., 2004; Atay and Karabacak, 2006). Interestingly, the capacity of cellular automata to carry out complex computations is not reduced, but rather *improved* when delays are present, similar e.g. to the improved synchronization that was found in other classes of delayed dynamical systems (Atay et al., 2004). Future research on DCAs will one the one hand focus on a more detailed understanding of their dynamical properties, but also on applications of this new class of dynamical systems to problems arising in distributed computation, and complex systems in nature and society.

References

F. M. Atay and O. Karabacak. Stability of coupled map networks with delays. *Siam Journal On Applied Dynamical Systems*, 5(3):508–527, 2006.

F. M. Atay, J. Jost, and A. Wende. Delays, connection topology, and synchronization of coupled chaotic maps. *Physical Review Letters*, 92(14), 2004.

E. Berlekamp, J. Conway, and R. Guy. *What is Life?*, chapter 25. Winning Ways for Your Mathematical Plays, Vol. 2: Games in Particular, 1982.

N. Fatès. Asynchronism induces second order phase transitions in elementary cellular automata. *Journal of Cellular Automata*, 4:21–38, 2009.

H. A. Gutowitz, J. D. Victor, and B. W. Knight. Local structure theory for cellular automata. *Physica D*, 18:48, 1987.

J. Jost. *Dynamical Systems: Examples of Complex Behaviour*. Universitext. Springer, 2005.

W. Li, N. H. Packard, and C. G. Langton. Transition phenomena in cellular automata rule space. *Physica D*, 45:77–94, 1990.

K. Lindgren and M. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.

C. Marr and M.-T. Hütt. Topology regulates pattern formation capacity of binay cellular automata on graphs. *Physica A*, 354:641–662, 2005.

T. Rohlf and C. Tsallis. Long-range memory elementary 1d cellular automata: Dynamics and nonextensivity. *Physica A*, 379:465–470, 2007.

S. Sakai, M. Kanno, and Y. Saito. Quiescent string dominance parameter f and classification of one-dimensional cellular automata. *Phys. Rev. E*, 69:066117, 2004. doi: 10.1103/PhysRevE.69.066117.

L. S. Schulman and P. E. Seiden. Statistical mechanics of a dynamical system based on conway's game of life. *J. Stat. Phys.*, 19:293–314, 1978.

J. M. G. Soto. Computation of explicit preimages in one-dimensional cellular automata applying the De Bruijn diagram. *Journal of Cellular Automata*, 3:219–230, 2008.

J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

S. Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55:601, 1983.

S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1, 1984.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 319–328

How do gliders move?

Emmanuel Sapin[1] and Olivier Sapin

[1] *Université de Rouen*

This paper deals with gliders in cellular automata. A study, based on gliders discovered by an evolutionary algortihm, identifies twelve different ways in which gliders move. The idea is to focus on a glider and on the new cells generated towards the direction of the glider. The classification of gliders is based on the neighbourhood of these cells at one generation before they appear.

Keywords: Gliders, Cellular Automata, Classification, Neighbourhoods, Cells

1 Introduction

The theories of complexity are the understanding of how independent agents are interacting in a system to influence each other and the whole system (1). A complex system can be described as a system composed of interconnected parts in which the whole exhibits more properties that the sum of the parts (2). Surprising computational tasks could result from interactions of independent agents in complex systems as emergence of computation is a hot topic in the science of complexity (3). A promising environment to study emergent computation is cellular automata (4) which are the simplest mathematical representation of complex systems (5) and an important modelling paradigm in the natural sciences and an extremely useful approach in the study of complex systems (6). They are uniform frameworks in which the simple agents are cells evolving through time on the basis of a local function, called the transition rules (7).

Emerging computation in cellular automata has different forms. Some have studied specific computation like density and synchronization tasks (8; 9) and pattern recognition (10). While others have considered *Turing-universal automata* (11; 12; 13) i.e. automata encompassing the whole computational power of the class of Turing machines (14). Some have wondered the question of the frequency of universal cellular automata as Wolfram (15). Some demonstrations of universality are based on mobile self-localized patterns of non-resting states (13), called *gliders* and these patterns are considered to be between order and chaos (16). The search for gliders is very competitive as it was notably explored by Adamatzky *et al.* with a phenomenological search (17), Wuensche who used his Z-parameter and entropy (18) and Eppstein (19). Lohn *et al.* (20) and Sapin *et al.* (21; 22; 23) have searched for gliders using evolutionary algorithms.

In this paper, a study, based on the first two thousand discovered gliders of the latter search, identifies twelve different ways in which gliders move. The paper is arranged as follows: the classification of gliders is described in Section 2 then the last section summarizes the presented results and discusses directions for future research.

Fig. 1: The seven possible neighbourhoods of the cell (c, y) at generation $n_c - 1$.

2 Classification of Gliders

Sapin *et al.* have searched for isotropic gliders using an evolutionary algorithm (21; 22). This algorithm found a lot of gliders accepted by different automata of a space of isotropic 2D 2-state automata using Moore neighbourhood. The study is based on the first two thousand gliders that were discovered on a run of this algorithm. These gliders are only orthogonal or diagonal as no oblique gliders were found.

Concerning orthogonal gliders, the idea is to locate a glider in a rectangle located at negative x-coordinate. Orthogonal isotropic gliders move toward the four cardinal points depending on their position therefore the shape of the glider is chosen in order to make it move towards the East. At some point, some cells of the glider will be at every positive x-coordinate. The first cell at each positive x-coordinate is taken into account. The idea is to try to determine what the neighbourhood of this cell was at the previous generation.

Concerning diagonal gliders, the idea is to locate a glider in a rectangle located at negative x-coordinate and at positive y-coordinate. Diagonal isotropic gliders move toward the four bisectrices depending on their position. Then the shape of the glider is chosen in order to make it move towards the South-East. At some point, some cells of the glider will be on every diagonal line with the equation $Y = X - c$ with c being a constant determining where the diagonal line crosses x-axis. The first cell on the diagonal line is taken into account. The idea is to try to determine what the neighbourhood of this cell was at the previous generation. The first subsection studies how do orthogonal gliders move. How do diagonal gliders move is studied in a second subsection.

2.1 Orthogonal Gliders

Let g be an orthogonal glider located to move towards the East. Let s_x and s_y be the size of the smallest rectangle containing the glider g at the generation 0. The glider is set up in a rectangle so that the upper right-hand corner is $(-1, -1)$ and the lower left-hand corner is $(-s_x, -s_y)$. It is possible to prove that for all $c > 0$ there exist generations at which one or more cells of state 1 will have c as an y-coordinate. Let n_c be the first generation when it happens and (c, y) the coordinates of a cell in state 1 such that no cell in state 1, with c as a x-coordinate and with an y-coordinate higher than y exists. At generation $n_c - 1$, as there is no cell in state 1 with c or $c + 1$ as x-coordinate, the neighbourhood of the point (c, y) can only be one of the seven neighbourhoods numbered figure 1. Let us consider all these neighbourhoods:

- If there exists c such that the neighbourhood of the cell (c, y) at generation $n_c - 1$ is the neighbourhoods 1 or 3, the neighbourhoods 1 and 3 are isotropic and a cell with the neighbourhoods 3 leads to the birth of a cell at every generation towards the North-East and a cell with the neighbourhoods 1 leads to the birth of a cell at every generation towards the South-East, this behaviors is not possible for any isotropic glider.

Fig. 2: Gliders of type α and period 1 at generations 0 and 1 separated by a comma.

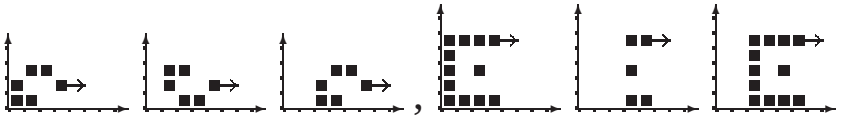


Fig. 3: Gliders of type α and period 2 at generations 0 and 1 separated by a comma.

- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is the neighbourhood 2, such gliders are called gliders of type α. The periods of all the discovered gliders of type α are 1, 2 and 3 and the velocity is 1. Figures 2, 3, 4 and 5 show the seven discovered gliders of this type. These figures and all others that show discovered gliders are created with the picture environment of latex and the code of these figures were generated automaticaly in part thanks to an analysis of the size, the period and the type of movement of every glider.

- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is the neighbourhood 6, two possibilities exist:

  – the cell (c, y) is in state 1 with only the other cell $(c, y - 1)$ in state 1 in the column c, such gliders are called gliders of type β_0 and 761 gliders of this type were found. The velocity of all the discovered gliders of type β_0 is 1 as shown in figures 6, 7, 8 and 9 for a sample of them with periods 1, 2, 3 and 4.

  – the cell (c, y) of the column c is in state 1 with a line of cells in state 1 from (c, y) to $(c, y - l)$ with $l > 1$, therefore the neighbourhood 7 allows also the appearence of cells at column c. The 146 discovered gliders of this type are called gliders of type β_1. The velocity of the discovered gliders of type β_1 is 1. Some of the discovered type β_1 gliders of periods 1, 2, 3 and 4 are shown in figures 10, 11, 12 and 13.

- If the neighbourhood of the cell (c, y) at generation $n_c - 1$ is the neighbourhood 4 then the cells $(c - 1, y + 1)$ and $(c - 1, y + 2)$ are in state 1, therefore there is a line of cells in state 1 from the cell $(c - 1, y + 1)$ to the cell $(c - 1, y + l)$ with $l > 1$. At generation $n_c - 1$ the cell $(c, y + l)$ has then the neighbourhood 6, the neighbourhoods 4 and 6 are isotropic, therefore at generation n_c the cell $(c, y + l)$ will be in state 1 but (c, y) are the coordinates of a cell in state 1 such that no cell in state 1, with c as a x-coordinate and with an y-coordinate higher than y exists. Therefore the neighbourhood of the cell (c, y) at generation $n_c - 1$ cannot be the neighbourhood 4.

- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is the neighbourhood 5, such

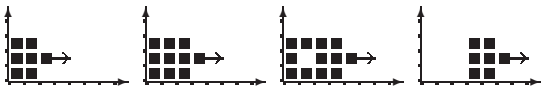


Fig. 4: A glider of type α and period 3 at generations 0,1,2 and 3.

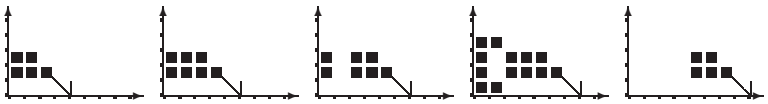


Fig. 5: The discovered glider of type α and period 4 at generations 0, 1, 2 and 3.

Fig. 6: Gliders of type β_0 and period 1 at generations 0 and 1 separated by a comma.

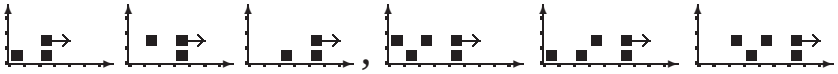


Fig. 7: Gliders of type β_0 and period 2 at generations 0, 1 and 2 separated by a comma.

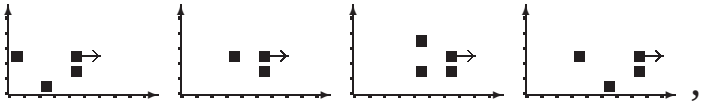


Fig. 8: A glider of type β_0 and period 3 at generations 0, 1, 2 and 3.

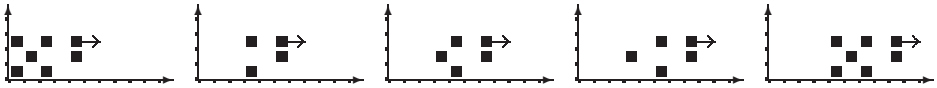


Fig. 9: A glider of type β_0 and period 4 at generations 0, 1, 2, 3 and 4.

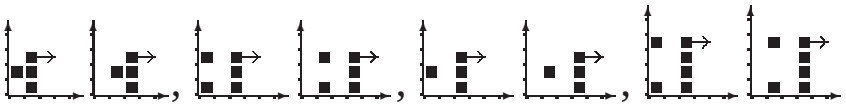


Fig. 10: Gliders of type β_1 and period 1 generations 0 and 1 separated by a comma.

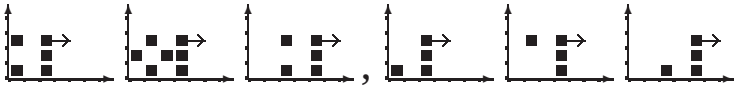


Fig. 11: Gliders of type β_1 and period 2 at generations 0, 1 and 2 separated by a comma.

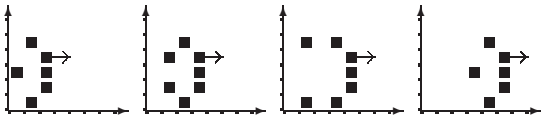


Fig. 12: A glider of type β_1 and period 3 at generations 0, 1, 2 and 3.

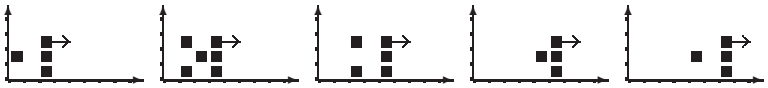


Fig. 13: A glider of type β_1 and period 4 at generations 0, 1, 2, 3 and 4.

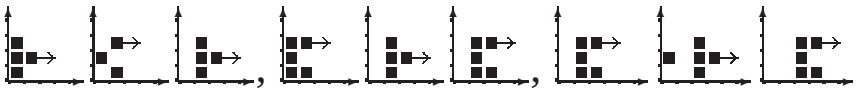


Fig. 14: Gliders of type γ_0 and period 2 at generations 0, 1 and 2 separated by a comma.

Fig. 15: Gliders of type γ_0 and period 3 at generations 0, 1, 2 and 3 separated by a comma.

gliders are called gliders of type γ_0. Among the two thousand discovered gliders, 278 gliders of type 20 were discovered. No gliders of type γ_0 were found of period 1. Figures 14, 15 and 16 show the discovered gliders of type γ_0 with periods 2,3 and 4.

- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is the neighbourhood 7 and at generation n the state of cells $(c, y + 1)$ and $(c, y - 1)$ is 0 then such gliders are called gliders of type γ_1. Among the two first thousand, there are 321 discovered gliders of type γ_1 and figures 17, 18 and 19 show a sample of them discovered of periods 2, 3 and 4.

- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is sometimes the neighbourhood 5, sometimes the neighbourhood 7 depending on c, such gliders are called gliders of type γ_2. Only three gliders of type γ_2 among the two thousand were discovered and they have a period of 4. One of these gliders is shown in figure 20.

There is no proof in this paper that gliders for which for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ cannot be among another set of neighbourhoods depending on c. Because such gliders were not found among the first two thousand gliders that were discovered, the conjecture that every glider is in only one of the six types α, β_0, β_1, γ_0, γ_1 and γ_2 is made.

2.2 Diagonal Gliders

Let g be a diagonal glider located to move towards the South-East. Let s_x and s_y be the size of the smallest rectangle containing the glider g at the generation 0. The glider is set up in a rectangle so that the upper right-hand corner is $(-1, -1)$ and the lower left-hand corner is $(-s_x, -s_y)$. For all c there exists generations at which one or more cells of coordinates $(X, X - c)$ is in state 1. Let n_c be the first generation when it happens and $(x, x - c)$ the coordinates of one of these cells such that no cell in state 1 of coordinates such that $Y = X - c$ and with an x-coordinate higher than x exists. The neighbourhood of the cell $(x, x - c)$ at generation $n_c - 1$ can only be one of the seven neighbourhoods numbered in figure 21. Let us consider all these neighbourhoods:

- If there exists c such that the neighbourhood of the cell $(x, x - c)$ at generation $n_c - 1$ is the neighbourhood 1, g cannot be a glider as shown in the section Orthogonal Gliders.

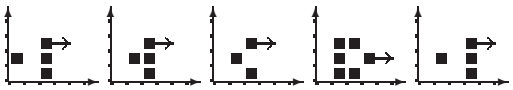


Fig. 16: A glider of type γ_0 and period 4 at generations 0, 1, 2, 3 and 4.

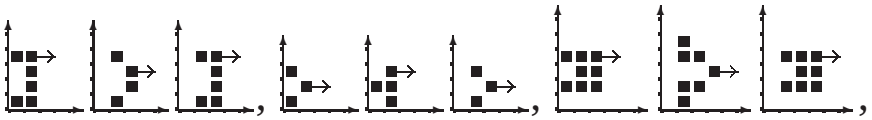


Fig. 17: Gliders of type γ_1 and period 2 at generations 0, 1 and 2 separated by a comma.

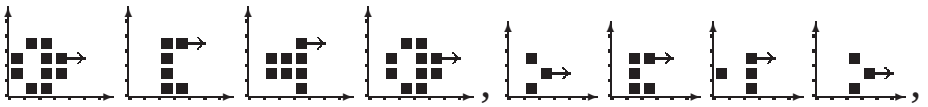


Fig. 18: Gliders of type γ_1 and period 3 at generations 0, 1, 2 and 3 separated by a comma.

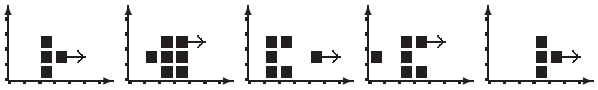


Fig. 19: A glider of type γ_1 and period 4 at generations 0, 1, 2, 3 and 4.

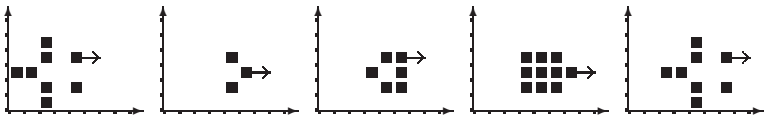


Fig. 20: A glider of type γ_2 and period 4 at generations 0, 1, 2, 3 and 4.

Fig. 21: The seven possible neighbourhoods of the cell $(x, x-c)$ at generation $n_c - 1$.

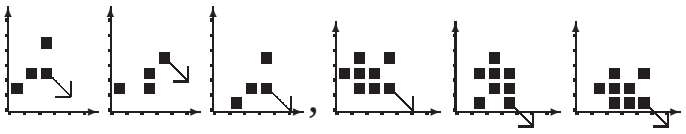


Fig. 22: Gliders of type ω_0 and period 2 at generations 0, 1 and 2.

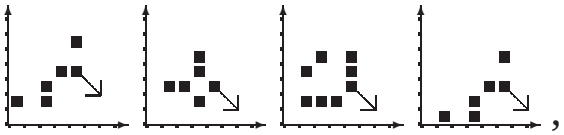


Fig. 23: Gliders of type ω_0 and period 3 at generations 0, 1, 2 and 3.

- It is conjectured that if there exists c such that the neighbourhood of the cell $(x, x-c)$ at generation $n_c - 1$ are the neighbourhoods 2 or 3 then g is an orthogonal glider so these neighbourhoods are not possible for the cell $(x, x-c)$ of a diagonal glider.
- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is sometimes the neighbourhood 4, sometimes the neighbourhood 5 depending on c, such gliders are called gliders of type ω_0. Only three gliders of type ω_0, shown in figures 22 and 23, among the two thousand were discovered.
- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is sometimes the neighbourhoods 4 or 5, sometimes the neighbourhood 6 or the neighbourhood 7 depending on c, such gliders are called gliders of type ω_1. Only one glider of type ω_1, shown in figure 24, among the two thousand was discovered and it has a period of 2.
- If for all c the neighbourhood of the cell (c, y) at generation $n_c - 1$ is sometimes the neighbourhoods 4 or 5, sometimes the neighbourhood 7 depending on c, such gliders are called gliders of type ω_2. Only three gliders of type ω_2 among the two thousand were discovered and they has a period of 2. Two of them are shown in figure 25.
- If for all c the neighbourhood of the cell $(x, x-c)$ at generation $n_c - 1$ is the neighbourhood 6, such gliders are called gliders of type ψ_0. 59 gliders of type ψ_0 were found among the two thousand gliders with periods of 3 and 4, as shown figures 26 and 27.
- If for all c the neighbourhood of the cell $(x, x-c)$ at generation $n_c - 1$ is the neighbourhood 7, such gliders are called gliders of type ψ_1. Some of these gliders with periods 3 and 4 are shown in figures 28 and 29.
- If for all c the neighbourhood of the cell $(x, x-c)$ at generation $n_c - 1$ is sometimes the neighbourhood 6, sometimes the neighbourhood 7 depending on c, such gliders, called gliders of type ψ_2, are shown figures 30 and 31.

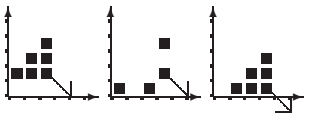


Fig. 24: The discovered glider of type ω_1 and period 2 at generations 0, 1 and 2.

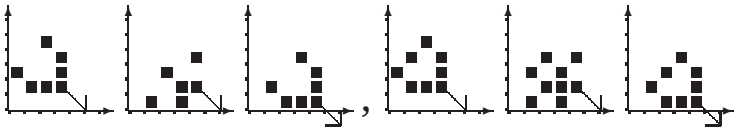


Fig. 25: Gliders of type ω_2 and period 2 at generations 0, 1 and 2.

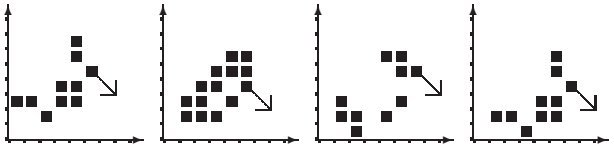


Fig. 26: Gliders of type ψ_0 and period 3 at generations 0, 1, 2 and 3.

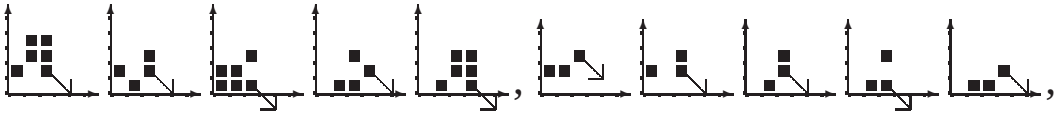


Fig. 27: Gliders of type ψ_0 and period 4 at generations 0, 1, 2, 3 and 4 separated by a comma.

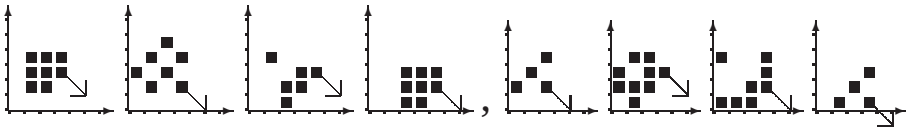


Fig. 28: Gliders of type ψ_1 and period 3 at generations 0, 1, 2 and 3 separated by a comma.

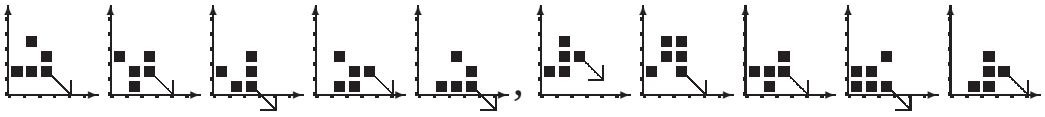


Fig. 29: Gliders of type ψ_1 and period 4 at generations 0, 1, 2, 3 and 4 separated by a comma.

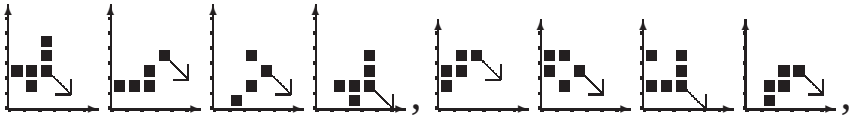


Fig. 30: Gliders of type ψ_2 and period 3 at generations 0, 1, 2 and 3 separated by a comma.

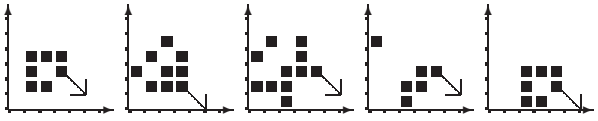


Fig. 31: A Glider of type ψ_2 and period 4 at generations 0, 1, 2, 3 and 4 separated by a comma.

The conjecture made for the neighbourhoods 2 or 3 implies every diagonal glider is in one of the six types ω_0, ω_1, ω_2, ψ_0, ψ_1 and ψ_2.

3 Synthesis and perspectives

This paper deals with the emergence of computation in complex systems with local interactions. A study about how gliders move is performed based on discovered gliders. Twelve types of movement have been identified depending on how new cells are generated in the sense of direction of the gliders. The first two thousands gliders discovered by the evolutionary algorithm of (21; 22; 24; 25; 26) have been automaticaly classified in one of the twelve types and some gliders of each type are shown in figures that were created with the picture environment of latex and the code of these figures were generated automaticaly in part thanks to an analysis of the size, the period and the type of movement of every glider..

The knowledge of the type of a glider can help to search for a glider gun emitting this glider and then to demonstrate the universality of automata that accept it. Moreover this research is a first step to lead to a better understanding of a link between the transition rules and gliders in cellular automata therefore a link between the emergence of computation in complex systems with simple components that is a new contribution to this theory of complex system. Future work could be to demonstrate the conjectures performed in this paper notably that only these twelve types exist or to find other types if any would exist. It also could be relevent to demonstrate properties of periods and velocities of some types. The study of type to movement could also be extended to gliders of automata with more than two states, gliders of hexagonal automata and gliders of automata with more than two dimensions.

References

[1] M.M. Waldrop. *Complexity: The Emerging Science at the Edge of Chaos. (New York: Simon and Schuster*. Simon and Schuster, New York, NY, 1992.

[2] Aristotle. *Metaphysics*. Book 8.6.1045a:8-10., (unknown).

[3] S. Wolfram. *A New Kind of Science*. Wolfram Media, Inc., Illinois, USA, 2002.

[4] J. Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, Ill., 1966.

[5] A. Ilachinski. *Cellular Automata*. World Scientific, 1992.

[6] G. Terrazas, P. Siepmann, G. Kendall, and N. Krasnogor. An evolutionary methodology for the automated design of cellular automaton-based complex systems. *Journal of Cellular Automata*, 2:77–102, 2007.

[7] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[8] M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos : Evolving cellular automate to perform computations. *Complex systems*, 7:89–130, 1993.

[9] M. Sipper. Evolution of parallel cellular machines. *D. Stauffer, editor, Annual Reviews of Computational Physics*, V. World Scientific:243–285, 1997.

[10] D. Wolz and P.B. de Oliveira. Very effective evolutionary techniques for searching cellular automata rule spaces. *Journal of Cellular Automata*, Vol 3, Issue 4:pp. 289–312, 2008.

[11] K. Morita, Y. Tojima, I. Katsunobo, and T. Ogiro. Universal computing in reversible and number-conserving two-dimensional cellular spaces. *In A. Adamatzky (ed.), Collision-Based Computing, Springer Verlag.*, pages 161–199, 2002.

[12] A. Adamatzky. Universal dymical computation in multi-dimensional excitable lattices. *International Journal of Theoretical Physics*, 37:3069–3108, 1998.

[13] P. Rendell. Turing universality in the game of life. *In Adamatzky, Andrew (ed.), Collision-Based Computing, Springer*, pages pp. 513–539, 2002.

[14] N. Ollinger. Universalities in cellular automata a (short) survey. *In B. Durand, editor, Symposium on Cellular Automata Journees Automates Cellulaires (JAC'08)*, pages pp. 102–118, 2008.

[15] S. Wolfram. Twenty problems in the theory of cellular automata. *Physica Scripta*, pages 170–183, 1985.

[16] A. Wuensche. Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the z parameter. *Complexity*, Vol 4, Issue 3:pp. 47–66, 1999.

[17] G. J. Martinez, A. Adamatzky, and H. V. McIntosh. Phenomenology of glider collisions in cellular automaton rule 54 and associated logical gates chaos. *Fractals and Solitons*, 28:100–111, 2006.

[18] A. Wuensche. Discrete dynamics lab (ddlab), `www.ddlab.org`, 2005.

[19] D. Eppstein. `http://www.ics.uci.edu/~eppstein/ca/`.

[20] J.D. Lohn and J.A. Reggia. Automatic discovery of self-replicating structures in cellular automata. *IEEE Transactions on Evolutionary Computation*, 1:165–178, 1997.

[21] E. Sapin, O. Bailleux, and J.J. Chabrier. Research of a cellular automaton simulating logic gates by evolutionary algorithms. *EuroGP03. Lecture Notes in Computer Science*, 2610:414–423, 2003.

[22] E. Sapin, O. Bailleux, J.J. Chabrier, and P. Collet. Demonstration of the universality of a new cellular automaton. *IJUC*, 2(3), 2006.

[23] E. Sapin, A. Adamatzky, and L. Bull. Searching for glider guns in cellular automata: Exploring evolutionary and other techniques. *EA07. Lecture Notes in Computer Science*, 4926:255–265, 2007.

[24] E. Sapin, A. Adamatzky, and L. Bull. Genetic approaches to search for computing patterns in cellular automata. *IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE*, 4(3):20–28, 2009.

[25] E. Sapin. Gliders and glider guns discovery in cellular automata. *In A. Adamatzky (ed.),Game of Life Cellular Automata, Springer Verlag.*, (in press).

[26] E. Sapin, A. Adamatzky, P. Collet, and L. Bull. Stochastic automated search methods in cellular automata: The discovery of tens of thousands glider guns. *Natural Computing*, (in press).

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 329–338

Stable Mixtures in Probabilistic Induction of CA Rules†

Burton Voorhees[1]

[1] *Center for Science*
Athabasca University
1 University Drive
Athabasca, AB
CANADA TT9S 3A3
burt@athabascau.ca

An induction algorithm provides unbiased best guess estimates of cellular automata rules generating time series of binary strings. If the strings were generated by a CA rule, the algorithm returns a rule in a symmetry class containing that rule. If the time series is random an unexpected outcome occurs: either the algorithm makes a type 1 error and still predicts a generating CA rule, or stable mixed choice cases appear in which the induction algorithm settles on a small set of rules as potential series generators. We present numerical results of these mixed choice outcomes, and an analytic explanation of how they are possible.

Keywords: caellular automata, stable mixtures

1 Introduction

In earlier papers [1, 2] a probabilistic induction algorithm was introduced, providing unbiased best guess estimates for the cellular automata rule generating a time series $\{\mu(t)\}$ of m-digit binary strings. The algorithm employs two probability distributions over the modeling set of elementary cellular automata rules, an a priori distribution and a choice distribution. The a priori distribution begins with conditions:

$$P(i,0) = \frac{1}{N}, \qquad \sum_{i=0}^{N-1} P(i,t) = 1 \tag{1}$$

where the modeling set contains N rules and $P(i,t)$ is the a priori probability for choice of rule R_i. At iteration t a single member of R is chosen, depending on $\mu(t)$ and the choice distribution $\{P^*(i,t)|0 \leq i \leq N-1\}$. Given this choice, the a priori distribution is updated by reinforcement. If element R_s from R was chosen at iteration t then

$$P(i,t+1) = \begin{cases} \frac{P(i,t)}{r} & i \neq s \\ \frac{r-1+P(i,t)}{r} & i = s \end{cases} \qquad 1 \leq r \tag{2}$$

†Supported by NSERC Discovery Grant OGP 0024871 and grants from the Athabasca University Research Committee.

The choice distribution is adapted to the predecessor profile of all rules in the modeling set R. This is necessary to eliminate bias, it would not do, for example, to allow prediction of a rule at any given iteration if that rule could not possibly have generated the strings given. If $n_i(\mu(t))$ is the number of $m+2$ digit predecessor strings of the string $\mu(t)$ for rule R_i then the choice at iteration t will be unbiased if the distribution used is

$$P^*(i,t) = \frac{n_i(\mu(t))P(i,t)}{\sum_{j=0}^{N-1} n_j(\mu(t))P(j,t)} \tag{3}$$

Once a choice is made, however, it is the a priori distribution $\{P(i,t)\}$ that is updated in accord with equation 2.

It might seem easier to test rules in the modeling class R to see if any of them satisfy the condition $R_i(\mu(t)) = \mu(t+1)$ for all t, but this method fails when $\{\mu(t)\}$ is the output of an apparatus that provides finite, discrete time measurements of a continuous dynamical system. Two considerations arise in this setting:

1. Spatial continuity implies that each $\mu(t)$ consists of the first m digits of a half infinite binary string giving the value of the continuous dynamical variable.

2. Temporal continuity means that the system will have passed through a continuum of states between the measured values $\mu(t)$ and $\mu(t+1)$.

For binary strings of length m, the predecessor profile $V(i)$ of a rule R_i is a vector in 2^m-dimensional Euclidian space defined by $V_\mu(i) = n_i(\mu_0 \ldots \mu_{m-1})$ where $\mu_0 \ldots \mu_{m-1}$ is the binary form of the index μ. Since all rules have the same total number of predecessors,

$$\sum_\mu V_\mu(i) = 2^{m+k+2} \tag{4}$$

and all profiles lie in a $2^m - 1$ dimensional simplex. The minimal length of the predecessor profile vector occurs for surjective rules, for which this vector terminates at the barycenter of the simplex. Since all strings have 4 predecessors for surjective rules, location in this simplex is determined by $v_\mu(i) = V_\mu(i) - 4$.

2 Empirical Results

If the series $\{\mu(t)\}$ was generated by a CA rule then, with high probability, the induction algorithm returned a rule in a symmetry equivalence class containing the generating rule. What is of interest, however, is the response of the induction algorithm when the series $\{\mu(t)\}$ is random. Two distinct outcomes are observed. The first and most frequent is that the algorithm still converges on a predicted series generator, committing an error of type 1. This is something that all induction is subject to so long as it is necessary to make a choice from a modeling class and there are no provisions that allow a response of "random." Indeed, it is well known in cognitive science that the brain itself is vulnerable to such errors and there are good evolutionary reasons that this is so. It is far safer to imagine a danger that is not present than it is to overlook a danger that is.

The second outcome is that a mixed choice, or "faceoff" appears, in which the algorithm settles into an apparently stable mixture of a small set of rules (2 – 5). What appears paradoxical is the stability of

these cases. If a rule is chosen at a given iteration, its a priori probability is reinforced. Thus, the initial expectation was that all faceoffs would be transients, eventually converging to a winning rule. This does not occur: faceoff cases appear within a few thousand iterations, and persist to the maximum number of iterations allowed (70,000 – 100,100). What occurs is that for certain combinations of rules, the fact that a rule is chosen at a given iteration leads to a decrease in its choice probability at the next iteration, even though the a priori probability is increased.

Analysis of faceoff data shows a correlation between rule table structure and faceoff frequency, both in terms of the λ and z parameters and in terms of rule decomposition into linear and non-linear parts. There is a correlation between the frequency that a rule appears in a faceoff and its predecessor profile, both for individual rules, and for rules showing up in faceoffs together. Figure 1 shows faceoff frequencies with $m = 8$ for the 256 elementary CA rules, ordered by increasing faceoff frequency, together with distance $||v||$ from the origin in the predecessor simplex. With the exception of the surjective rules, for which $||v|| = 0$ and the faceoff frequency is close to 0, this figure indicates an inverse relation between $||v||$ and faceoff frequency. The exception for surjective rules occurs because a surjective rule will win against any non-surjective rule—the few faceoffs that occur involving surjective rules only involve such rules. It also appears that faceoffs involving rules from the same symmetry class are rare, and that faceoffs tend to involve rules separated by an intermediate distance in the predecessor profile simplex.

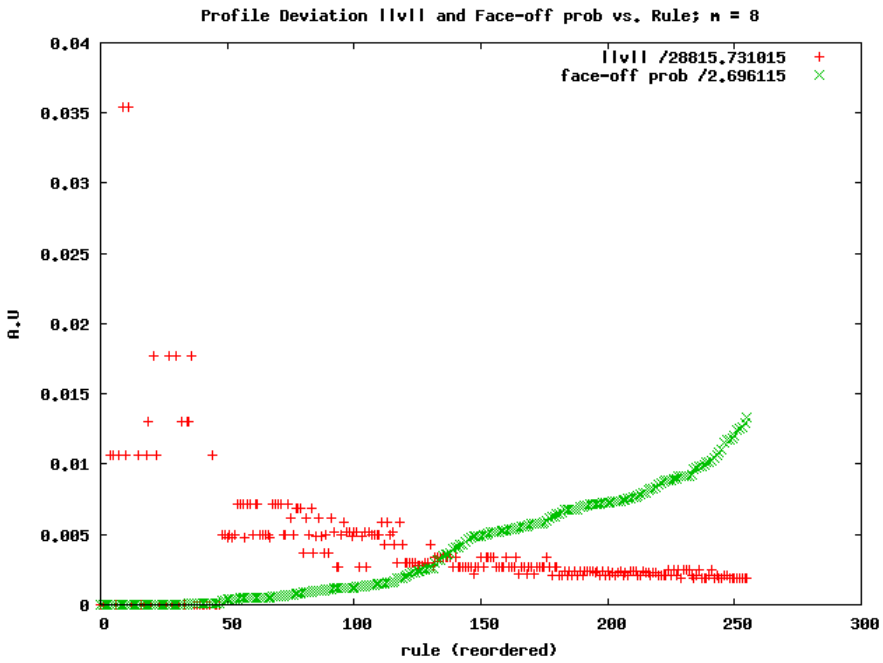


Fig. 1: Profile Deviation $||v||$ and Faceoff Frequency for 256 Elementary Rules on Strings of Length 8 (Note: Rules Ordered by Increasing Faceoff Frequency)

There is some evidence of a power law relation between faceoff frequencies and the value of $||v||$. Plots

of $log||v||$ versus the log of faceoff frequency f for $3 \leq m \leq 8$ for all r values show a value for the slope of the linear regression of $-.323674 \pm .009980$, leading to a relation $f \sim K||v||^{-3.089553}$. Taking the standard deviation of the slope into account, this suggests an inverse cube relation. This breaks down, however, when data for varying r is examined. As shown in Table 1, this data indicates a dependence on both m and r. For $r = 1.01$ the linear relation remains for all m, but now the slope varies as indicated in Table 1. For $r = 1.05$ a reasonable linear relation only appears for m greater than five, while for $r = 1.1$ it only shows up at $m = 8$ and not at all for $r = 1.2$ or greater.

m	3	4	5	6	7	8
r=1.01	-.362621	-.2163944	-.189907	-.183785	-.174483	-.153609
r=1.05	Linear relation unclear			-.293239	-.321353	-.331769
r=1.1	No discernable linear relation					-.339981
r=1.2	No discernable linear relation					

Tab. 1: Slope of Linear Regression for log$||v||$ vs. log(f) for m = 3,. . . ,8

Another perspective is provided by Figure 2, which shows faceoff frequencies for the top 700 pairs of rules occurring in faceoffs for $r = 1.01$ with $m = 8$, combined with the distance between rules found in these pairs. Also indicated in this figure are the values of this distance for rules related by the symmetry operations of neighborhood reversal (T_1) and predecessor reversal (T_3), and the expected distance of rule pairs chosen at random. Plots of similar figures for values of m from 3 to 7 shows the distance between rules in faceoffs gradually increasing toward the random expectation line indicating that there is an effect due to string length. The faceoff distances do not obviously fall below what would be expected on a random distribution until $m = 5$. Another effect that shows up when all faceoffs are considered rather than only the most frequent 700 is that the distance distribution remains relatively flat until about the 2500-th pair and then suddenly increases. This is indicated in Figure 3. No explanation has been found as yet for the apparent quantization of distances that appears with the sharp jump.

The apparent stability of mixed choice responses was tested by forcing the occurrence of faceoffs. This was done by choosing two rules and artificially setting their initial probabilities between .49 and .50. The induction algorithm was then continued for $100,100$ iterations following the emergence of a faceoff. Data on a priori probabilities was taken at every 100 iterations of the induction algorithm yielding 1001 data points for each rule in the faceoff. Mean a priori probabilities were computed together with their standard deviations. Table 2 shows the results for four 2-rule and four 3-rule faceoffs.

3 Faceoff Stability

Stability of faceoffs arises through details of the induction algorithm, which involves both the a priori probabilities of equations 1 and 2, and contributions from the predecessor profiles of all rules in the modeling class in the choice probabilities of equation 3. This produces cases in which choice of the random string $\mu(t + 1)$ results in the probability that a rule is chosen at iteration $t + 1$ being less than its probability of being chosen at iteration t, even though it *was* chosen at t so that it's a priori probability is increased for iteration $t + 1$. The obvious case is if the string $\mu(t + 1)$ is a Garden-of-Eden string for the rule chosen at t. The next theorem describes the more general case.

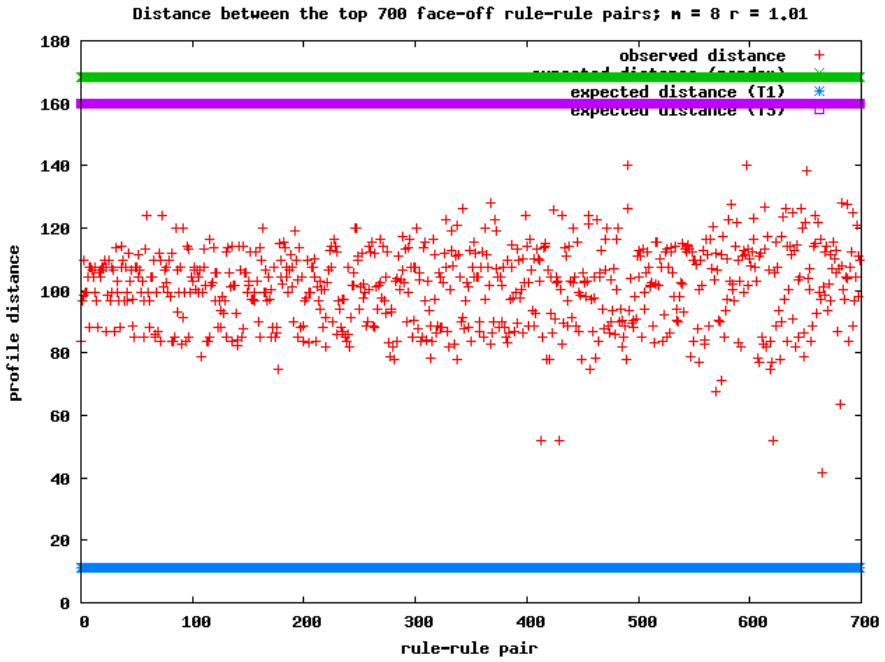


Fig. 2: Distances Between Top 700 Rule Pairs in Faceoffs ($r = 1.01$, $m = 8$) (Red shows distances between faceoff pairs, Blue line shows expected distance for T_1 related rules, violet for T_3 related rules, green for randomly related rules)

Rules in Faceoff	Onset	Mean Frequency at Iteration	Standard Deviations
(152,229)	5400	(.51257, .48742)	(.05174, .05174)
(57,218)	64,500	(.45920, .54079)	(.05411, .05411)
(161,122)	6800	(.50676, .49323)	(.05341, .05340)
(145,173)	2900	(.44342, .55658)	(.05061, .05061)
(12,58,124)	3200	(.13435, .38054, .48510)	(.02990, .05313, .05059)
(88,126,159)	2900	(.67616, .11973, .20410)	(.04051, .03671, .04189)
(96,182,244)	11,700	(.25799, .58103, .16097)	(.03893, .04796, .04325)
(28,31,39)	2300	(.36764, .29421, .33815)	(.04364, .05123, .05591)

Tab. 2: Mean a Priori Probabilities and Standard Deviations for Some Forced Faceoffs

Theorem 1 *If a rule R_i is chosen at iteration t, the condition for $P^*(i,t+1) - P^*(i,t) < 0$ is*

$$\frac{P(i,t)}{r-1+P(i,t)} > \frac{n_i(\mu(t+1))n_j(\mu(t))}{n_i(\mu(t))n_j(\mu(t+1))} \tag{5}$$

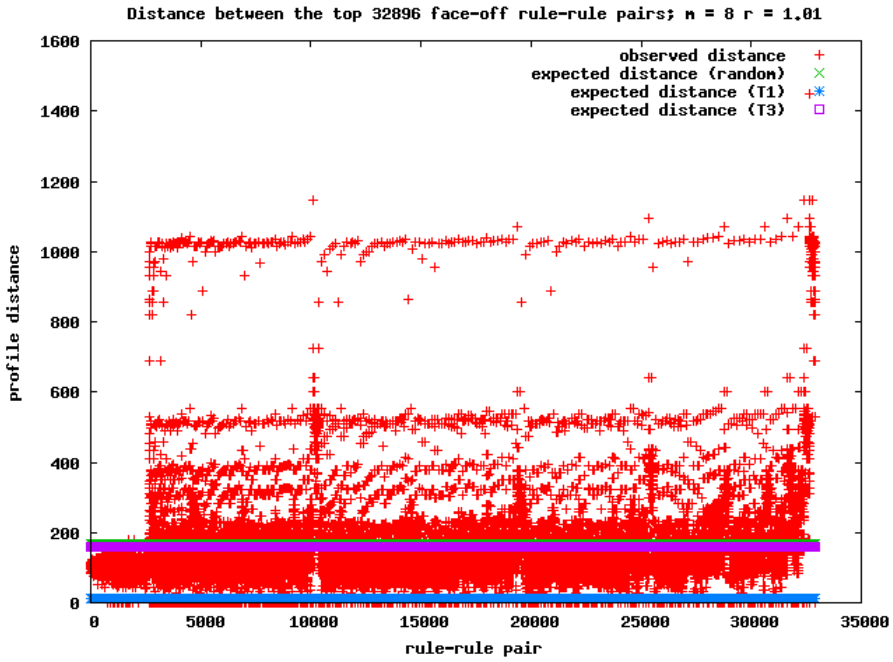


Fig. 3: Distance between $32,896$ rule pairs occurring in faceoffs.

Table 3 shows the percent of pairs of eight digit strings satisfying the condition of equation 5 with $r = 1.01$, for the four two rule faceoffs of Table 2 and for two comparison cases of rule pairs $(18, 90)$ and $(24, 126)$, which have never appeared together in faceoffs. The satisfaction percentage indicates the percent of all pairs of eight digit strings $(\mu(t), \mu(t+1))$ with $n_i(\mu(t)) \neq 0$ that satisfy equation 5. The larger it is, the more likely equation 5 is to be satisfied, meaning that rule R_i is less likely to be chosen on the iteration following the one at which it was chosen. The $n_i(\mu(t)) \neq 0$ condition is imposed since by assumption rule R_i was chosen at iteration t and this could not have occurred if $n_i(\mu(t)) = 0$. Note that if $n_j(\mu(t+1)) = 0$ equation 5 can never be satisfied.

Rule Pair	Satisfaction Percentage
(152,229)	(.53414, .57045)
(57,218)	(.54645, .46960)
(161,122)	(.53375, .53375)
(145,173)	(.54820, .56682)
(18,90)	(.68640, .38878)
(24,126)	(.44453, .21447)

Tab. 3: Satisfaction Percentages For Selected Rule Pairs in 2-Rule Faceoffs and Comparison Pairs

The first comparison case involves rules 18 and 90 (a surjective rule). In this case, examination of the satisfaction percentages shows a strong bias for continued choice of rule 90. In the second case, neither satisfaction percentage is above .5, indicating that the two rule situation is unstable.

In three of the four faceoff cases shown, both rules have satisfaction percentages greater than .5, which can be taken as indication of stability. The remaining case involves rules 57 and 218. This case is also unusual in that the onset of the two rule faceoff only occurs at iteration $64,500$ and it appears to emerge from the collapse of a three rule faceoff. Since a cutoff on probability was used to determine which rules to include in faceoffs, this case may represent a situation in which a three rule faceoff has one rule with a very low probability, or possibly a two rule case in which additional stability is provided by an infrequent invasion of other rules.

While Theorem 1 involves a specific pair of strings $\mu(t)$ and $\mu(t+1)$, it is also possible to obtain a stability condition that is independent of the random string generated at any given iteration. Let $\Delta(i|t,t+1) = \langle P^*(i,t+1)\rangle_\mu - \langle P^*(i,t)\rangle_\mu$ where $\langle\cdot\rangle_\mu$ indicates the average computed over all m-digit strings. If $E[\Delta(i|t,t+1)]$ is the expected value of $\Delta(i|t,t+1)$, the choice probability of rule R_i will be expected to increase or decrease with continued iteration of the induction algorithm as $E[\Delta(i|t,t+1)]$ is respectively greater than or less than 0. The condition for rule R_i to be in a stable faceoff is that $E[\Delta(i|t,t+1)] = 0$.

Theorem 2 *Let $E[\Delta(i|t,t+1)]$ be the expectation value of $\Delta(i|t,t+1)$. The condition for $E[\Delta(i|t,t+1)] = 0$ is*

$$P(i,t)\sum_{s\neq i} P(s,t)\left[\sum_{\mu}\frac{n_i(\mu)-n_s(\mu)}{v(\mu,t)}\right]\left[\sum_{\mu'}\frac{n_i(\mu')-n_s(\mu')}{D(\mu',t)}\right] = 0 \tag{6}$$

where

$$v(\mu,t) = \sum_s n_s(\mu)P(s,t), \qquad D(\mu',t) = \frac{1}{r}v(\mu',t)[v(\mu',t)+r-1] \tag{7}$$

If a faceoff is to be metastable, the condition $E[\Delta(i|t,t+1)] = 0$ must hold for each rule involved in the faceoff. If only two rules are involved, say R_i and R_j then, from equation 9, it is necessary that

$$d_{ij}(t) = \sum_{\mu}\frac{n_i(\mu)-n_j(\mu)}{v(\mu,t)} = 0 \tag{8}$$

Note that this equation is identically satisfied for surjective rules. Table 4 gives the values of $d_{ij}(t)$ for the 2-rule faceoffs of Table 3, averaged over the$100,100$ iterations of the induction algorithm, as sampled every 100 iterations, as well as for the comparison cases of rules $(18,90)$ and $(24,126)$. While all of the faceoff cases are well within their standard deviation of 0, the comparison values are large (since the comparison rules are never involved in faceoffs, the value was computed on the assumption that the probability for both rules was .5. Hence there is no standard deviation for these cases).

Rules	(152,229)	(57,218)	(161,122)	(145,173)	(18,90)	(24,126)
$\langle d_{ij}(t)\rangle_t$	-.53572	2.16914	2.99782	-.25306	-228.60807	-111.82395
Standard Deviation	25.24048	20.77144	23.98559	28.22384		

Tab. 4: $d_{ij}(t)$ Averaged Over t Together With Standard Deviation

Examination of equation 6 shows how the presence of an additional rule in a faceoff can help to stabilize it. For a three rule faceoff involving rules R_i, R_j, and R_k the expectation values to consider are:

$$
\begin{aligned}
E[\Delta(i|t,t+1)] &= P(i,t)\left[P(j,t)\Phi_{ij}(t)\sum_{\mu}\frac{n_i(\mu)-n_j(\mu)}{v(\mu,t)} + P(k,t)\Phi_{ik}(t)\sum_{\mu}\frac{n_i(\mu)-n_k(\mu)}{v(\mu,t)}\right] \\
E[\Delta(j|t,t+1)] &= P(j,t)\left[P(i,t)\Phi_{ij}(t)\sum_{\mu}\frac{n_j(\mu)-n_i(\mu)}{v(\mu,t)} + P(k,t)\Phi_{jk}(t)\sum_{\mu}\frac{n_j(\mu)-n_k(\mu)}{v(\mu,t)}\right] \\
E[\Delta(k|t,t+1)] &= P(k,t)\left[P(j,t)\Phi_{jk}(t)\sum_{\mu}\frac{n_k(\mu)-n_j(\mu)}{v(\mu,t)} + P(i,t)\Phi_{ik}(t)\sum_{\mu}\frac{n_k(\mu)-n_i(\mu)}{v(\mu,t)}\right] \\
\Phi_{ij}(t) &= \sum_{\mu}\frac{n_i(\mu)n_j(\mu)}{D(\mu,t)}
\end{aligned}
\tag{9}
$$

Note that $E[\Delta(i|t,t+1)]+E[\Delta(j|t,t+1)]+E[\Delta(k|t,t+1)]=0$, reflecting conservation of probability, and that when $P(k,t)=0$ this reduces to the case of equation 8. Now consider a case for which

$$
\sum_{\mu}\frac{n_i(\mu)-n_j(\mu)}{v(\mu,t)}>0, \qquad \sum_{\mu}\frac{n_j(\mu)-n_k(\mu)}{v(\mu,t)}>0, \qquad \sum_{\mu}\frac{n_k(\mu)-n_i(\mu)}{v(\mu,t)}>0 \tag{10}
$$

No pair of rules satisfies equation 7 so no two-rule faceoff involving these rules can be stable. Nevertheless, each of the expectation values in equation 9 can be zero if the bracketed terms in equation 9 are all zero. In matrix form this is expressed as:

$$
\begin{bmatrix}
1 & -\frac{\Phi_{jk}(t)d_{jk}(t)}{\Phi_{ik}(t)d_{ki}(t)} & 0 \\
0 & 1 & -\frac{\Phi_{ik}(t)d_{ki}(t)}{\Phi_{ij}(t)d_{ij}(t)} \\
-\frac{\Phi_{ij}(t)d_{ij}(t)}{\Phi_{jk}(t)d_{jk}(t)} & 0 & 1
\end{bmatrix}
\begin{bmatrix} P(i,t) \\ P(j,t) \\ P(k,t) \end{bmatrix} = 0 \tag{11}
$$

It is easy to show that the eigenvalues of this matrix are just the cube roots of unity, illustrating the cyclic rock-paper-scissors nature of the three rules involved.

4 Discussion

The apparent stability of mixed choice induction suggests a more general possibility. In inductive networks with established modeling classes of patterned responses, stable higher-level response patterns may emerge as stochastic mixed choice blends. This provides a mechanism of behavioral emergence as well as offering a potential selective advantage in cases in which response to ambiguous input is required. This also provides a mechanism for the emergence of mixed response strategies that succeed even though each individual strategy in the blend fails if utilized alone—a situation arising in paradoxical games [3, 4, 5].

5 Acknowledgements

This work was carried out with the assistance of NSERC Undergraduate Summer Research Assistants Todd Keeler (2004, 2005) and Rhyan Arthur (2003, 2005) and supported by NSERC Discovery Grant OGP 0024871.

References

[1] Voorhees, B., Arthur, R., and Keeler, T. Probabilistic induction of cellular automata rules: I. A reinforcement scheme. International Journal of Unconventional Computing **2**(2) (2006) 91 – 127.

[2] Voorhees, B., Arthur, R., and Keeler, T. Probabilistic induction of cellular automata rules: II. Probing CA rule space. International Journal of Unconventional Computing **2**(3) (2006) 195 – 229.

[3] Martin, H., and von Baeyer, H.C. Simple games to illustrate Parrondos paradox. American Journal of Physics **72** (2004) 710 – 714.

[4] Harmer, G.P., and Abbott, D. Losing strategies can win by Parrondos paradox. Nature **402** (1999) 864.

[5] Behrends, E. On Astumians paradox. Fluctuation and Noise Letters **5**(1) (2005) L109 – L125.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 339–350

A Note on (Intrinsically?) Universal Asynchronous Cellular Automata

Thomas Worsch[1]

[1] *Faculty of Informatics*
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

We consider asynchronous one-dimensional cellular automata (CA). It is shown that there is one with von Neumann neighborhood of radius 1 which can simulate each asynchronous one-dimensional cellular automaton. Analogously all α-asynchronous CA (where each cell independently enters a new state with probability α) can be simulated by one α-asynchronous CA (with the same probability for state updates) with von Neumann neighborhood of radius 1.

We also point out a few open problems for asynchronous CA.

Keywords: cellular automata, intrinsic universality, asynchronous updating

1 Introduction

Asynchronous cellular automata (ACA) are cellular automata where in each global step only the cells in an arbitrary (non-empty) subset of all cells make a state transition while the others retain their current states. Recently so-called α-asynchronous CA have gained a lot of interest. Here $\alpha \in]0; 1]$ is the probability with which each cell independently of the others makes a state transition during a global step.

This paper is organized as follows: In Section 2 we review the basic definitions and the known constructions for asynchronous CA as far as they are relevant in the present context. The core of the paper is Section 3 where the overall construction is sketched and the main technical tool explained which give rise to the following result:

Theorem 1 *There is a purely asynchronous deterministic CA which is able to simulate all purely asynchronous deterministic CA.*

The precise definition of "simulation" needed in the theorem can be seen from the constructions below. We consider this to be a reasonable approach, but admittedly the situation is more complicated then in the synchronous deterministic case. In addition until a satisfying definition is agreed upon we use the adverb "intrinsically" informally: For a set of automata $\mathcal{M}$ a member $M \in \mathcal{M}$ is intrinsically universal for $\mathcal{M}$, if it can simulate each $M' \in \mathcal{M}$.

It should be pointed out, that an extension of the precise notion of intrinsic universality for S-DCA, as proposed e. g. by Ollinger (2008), to CA which are *not* S-DCA is still missing. This is the reason why

the word "intrinsically" has been put in parenthesis and furnished with a question mark in the title of this paper.

The main technical problem is that on one hand despite asynchronous updating the universal simulator has to work reasonably (this is easy) while on the other hand one has to exploit the asynchronicity of the simulator to generate the different possibilities for global steps of the simulated asynchronous CA.

A closer inspection of the construction reveals that the same ideas can be one applied in several other situations. These possibilities are discussed in Section 4. We conclude with a summary and a short outlook in Section 5.

2 Basics

2.1 General notation

In this paper we are interested in one-dimensional cellular automata. If the set of states of one cell is denoted as Q, the set of all configurations is $Q^{\mathbb{Z}}$. (We write B^A for the set of all functions from A to B.) A neighborhood is a finite set $N = \{\nu_1, \dots, \nu_k\}$ of integers. A local configuration is a mapping $\ell : N \to Q$; thus Q^N is the set of all local configurations. The local configuration c_{i+N} observed by cell $i \in \mathbb{Z}$ in the global configuration c is defined as $c_{i+N} : N \to Q : n \mapsto c(i+n)$.

The behavior of a single cell of a nondeterministic CA (NCA) is described by the local transition function $f : Q^N \to 2^Q$. (We write 2^M for the powerset of M.) An NCA is a deterministic CA (DCA) iff for all $\ell \in Q^N$ holds: $|f(\ell)| = 1$. For a probabilistic CA (PCA) the local transition function is of the form $p : Q^N \to [0;1]^Q$, where $p(\ell)(q)$ is the probability that a cell enters state q if it observes ℓ in its neighborhood. For PCA it is required that for all $\ell \in Q^N$ the sum $\sum_{q \in Q} p(\ell)(q) = 1$. To each PCA there is a corresponding NCA with local transition function $f : Q^N \to 2^Q : \ell \mapsto \{q \mid p(\ell)(q) > 0\}$. Whenever we speak about PCA and use some notation for NCA, we mean the corresponding NCA as just defined.

We call each tuple $(q_1, \dots, q_k, q')$ with $q' \in f(q_1, \dots, q_k)$ a *rule* of the CA.

The triple (Q, f, N) is called the *local structure* of a CA.

2.2 Updating schemes

In general a local structure (Q, f, N) together with a prescription how cells are updated induce a global transition relation $F \subseteq Q^{\mathbb{Z}} \times Q^{\mathbb{Z}}$ describing the possible global steps. If $(c, c') \in F$ we will also write $c \vdash c'$ (possibly with an index for further clarification). In a global step each cell has two possibilities: to be *active* and make a state transition (according to a rule) or to be *passive* and not to change its state. Restrictions made by different updating schemes lead to different possible behaviors of CA.

A (finite or infinite) sequence $(c_0, c_1, c_2, \dots)$ of configurations is a *computation*, iff for all pairs (c_i, c_{i+1}) within the sequence it is true that $c_i \vdash c_{i+1}$.

Synchronous updating. Synchronous updating means that in a global step all cells are active. Hence for an NCA $c \vdash^s c'$ holds iff $\forall i \in \mathbb{Z} : c'(i) \in f(c_{i+N})$. Of course, for deterministic CA the global step relation F is in fact a function. We will use the prefix S- to indicate synchronous updating (S-NCA etc.).

Now, we'll have a look at different types of asynchronous updating.

Purely asynchronous updating. The first version of asynchronous updating has been considered for many years now (see e. g. Nakamura, 1974). In order to distinguish it from the other forms mentioned

below we call this version *purely asynchronous* updating. In this case in each global step there are no restrictions on whether a cell may be active or passive. Thus for an NCA $c \vdash^a c'$ holds iff $\forall i \in \mathbb{Z} : c'(i) \in f(c_{i+N}) \vee c'(i) = c(i)$. (We remark that formally it is allowed that *no* cell is active in a global step. But nothing in this paper gets wrong, if one requires that in each global step the set of active cells is non-empty.)

Obviously, even for deterministic CA purely asynchronous updating can lead to many different possible computations starting with the same configuration. See Section 2.3 below for further remarks on this.

We will use the prefix A- to indicate asynchronous updating (A-NCA etc.) and use the term *asynchronous CA* (ACA) for A-DCA (with a deterministic local function!).

α-asynchronous updating. In recent years so-called α-asynchronous CA have attracted some attention. Here, $\alpha \in]0;1]$ is a positive probability. Similar to PCA one considers the behavior (active or passive) of each cell during a global step as a random variable, and α is the (uniform) probability of a cell to be active. We will write $\beta = 1 - \alpha$ for the probability that a cell remains passive.

We will use the prefix A(α)- to indicate α-asynchronous updating (e. g. A(0.5)-NCA etc.).

Fully asynchronous updating. In the fully asynchronous updating scheme it is required that in each global step only exactly one cell is active. We write $c \vdash^{fa} c'$ iff there is a cell $i \in \mathbb{Z}$ such that $c'(i) \in f(c_{i+N})$ and $\forall j \neq i : c'(j) = c(j)$. Even for relatively simple DCA (e. g. the elementary DCA or two-dimensional minority) the analysis of their behavior under fully asynchronous updating is surprisingly "nonsimple" Fatès and Gerin (2008); Regnault, Schabanel, and Thierry (2009); Lee, Adachi, Peper, and Morita (2004).

2.3 Relations between different types

We will now review some known relations between different types of CA. This quickly leads to the notion of *simulation*. In the following we will speak about *guest CA* and *guest cells* and about *host CA* and *host cells*. The host is the simulating CA and the guest is the simulated CA.

2.3.1 The obvious

It should be clear that ACA, i. e. A-DCA, are a special case of S-NCA in the following sense: Assume that A is an ACA with local structure (Q, f_A, N_A). Define an NCA B with local structure (Q, f_B, N_B) as follows: the set of states is the same and the neighborhood is $N_B = N_A \cup \{0\}$ (may be the same, too). For each local configuration $\ell : N_B \to Q$ one requires $f_B(\ell) = \{\ell(0)\} \cup f_A(\ell|_{N_A})$. (Here we use the notation $f|_M$ for the restriction of function f to the subset M of its domain.)

Then $\vdash_A^a$ is the same as $\vdash_B^s$. This is so, because given $c \vdash_A^a c'$, a cell i in configuration c' of A has the possibilities $c(i)$ and $f_A(c_{i+N_A})$ (by definition of asynchronicity); and given $c \vdash_A^a c'$ in configuration c' a cell i in B has the possibilities $c(i)$ and $f_A(c_{i+N_A})$ (by definition of f_B) as well.

So in a very strong sense each A-DCA A can be simulated by a S-NCA B: the induced global step relations are exactly the same. In general the reverse simulation, in the same sense, of S-NCA by A-DCA is impossible since a cell of an S-NCA may enter one of three or more different states while a cell of an A-DCA has at most two choices.

Analogously one can consider A(α)-DCA as a special case of PCA, but not vice versa.

2.3.2 Golze's construction.

Golze (1978) has shown how for each S-NCA B one can construct an A-DCA A simulating B. Besides

the obstacle of different numbers of choices just mentioned, there is another problem. Whenever one uses some kind of asynchronous updating, there are infinite computations in which only a constant number of cells is ever active. Such computations are not useful at all. Roughly speaking, the solution proposed by Golze (1978) is to consider equivalence classes of space-time diagrams where for example computations as just mentioned are equivalent to the trivial computation where nothing at all has happened.

The simulations described in Section 3 are reasonable in the sense that the overall approach is along the lines already proposed by several authors.

It should be noted that in Golze's construction the size of the neighborhood of the host depends on the maximum number of nondeterministic choices in one local situation and cannot be bounded by a constant. As we will point later in some more detail, the construction in Section 3 can be used to achieve the same while only using von Neumann neighborhood of radius 1 in *all* cases.

2.3.3 Nakamura's construction.

Nakamura (1974) has described how an S-DCA D can be simulated (again in a specific sense) by an A-DCA A. The problem to overcome is that uncontrolled active state changes of one cell may lead to totally "irrelevant" configurations if neighboring cells do not become active at all. We briefly sketch the idea (citing from a paper by Worsch and Nishio (2009)).

As the set of states for A one uses $Q_A = Q_D \times Q_D \times \{0,1,2\}$. Let c^t denote the configuration reached by D after t steps from some initial configuration c. If in a given configuration c_A of A cell j of A has already simulated t transitions of cell j of S then $c_A(j) = (c^t(j), c^{t-1}(j), t \bmod 3)$. Therefore we denote by $current(q)$, $old(q)$, and $time(q)$ the first, second, and third component of a state $q \in Q_A$ respectively. $time(q)$ is also called the time stamp of the cell.

In order to maintain this invariant, given $q_1, \ldots, q_k$ the local function $f_A(q_1, \ldots, q_k)$ is defined as follows, assuming without loss of generality that $\nu_1 = 0$:

- If for all i: $time(q_i) = time(q_1)$ or $time(q_i) = time(q_1) + 1 \pmod 3$, then $f_A(q_1, \ldots, q_k) = (f_D(q'_1, \ldots, q'_k), current(q_1), time(q_1) + 1 \pmod 3)$, where

$$q'_i = \begin{cases} current(q_i) & \text{if } time(q_i) = time(q_1) \\ old(q_i) & \text{if } time(q_i) = time(q_1) + 1 \pmod 3 \end{cases}$$

- otherwise $f_A(q_1, \ldots, q_k) = q_1$.

If a cell is updated according to the first alternative, we will say, that it *makes progress*.

As in Golze's construction also in this case for each guest CA to be simulated another host CA to simulate is used. What we will describe in the following two sections is *one host* being able to simulate *all guests* from an infinite set of CA.

3 Universal simulation of purely asynchronous DCA

In this section we will describe the construction of a purely asynchronous DCA able to simulate each purely asynchronous DCA. The cases of α-asynchronous and fully asynchronous updating will be discussed in Section 4.

Since we are interested in one host being able to simulate different guests for different initial configurations it is necessary to provide the host with an encoding of the local structure of the guest and an encoding

of the initial guest configuration. These are described in Sections 3.1 and 3.2. The general structure of the simulation is outlined in Section 3.3. This will be done in such a way that it can not only be used in the purely asynchronous setting, but for α-asynchronous updating as well.

The description will make use of so-called *asynchronous coins*. These are black boxes consisting of two adjacent cells. A toss of the coin can be requested by a signal from outside and the result will be a `0` or a `1` (once the cells have been active a constant number of times). In Section 3.3.4 it will be explained how one can choose the local transition function for the two cells of an asynchronous coin in a purely asynchronous host. This is the only detail which has to be modified slightly for α-asynchronous CA in Section 4.

3.1 Encodings of local structures

For convenience we will use the alphabet $\{\texttt{0}, \texttt{1}, \texttt{[}, \texttt{]}\}$ for representing all the pieces of a guest CA on the host. Without loss of generality $Q = \{0, \ldots, n-1\}$. The encoding of a single guest state is $cod_Q(q) = \texttt{[}\, bin(q) \texttt{]}$ where $bin(q) \in \{\texttt{0}, \texttt{1}\}^+$ is the binary representation of q, all of them having the same length $\lceil \log_2 |Q| \rceil$.

As the encoding of a single local rule $(q_1, \ldots, q_k, q')$ of a CA we use $\texttt{[}\, cod_Q(q_1) \cdots cod_Q(q_k)\, cod_Q(q') \texttt{]}$. The whole local transition function is encoded as $cod(f) = \texttt{[}\langle$*concatenation of encodings of all local rules*$\rangle \texttt{]}$.

The members of the neighborhood can for example be encoded as $cod_N(\nu_i) = \texttt{[1[}\, bin(-\nu_i) \texttt{]]}$ if $\nu_i \in N$ is negative and as $cod_N(\nu_i) = \texttt{[0[}\, bin(\nu_i) \texttt{]]}$ if $\nu_i \in N$ is non-negative. That allows to find out easily whether a neighbor is to the left or to the right and how far. The complete neighborhood is encoded as the word $cod(N) = \texttt{[}\, cod_N(\nu_1) \cdots cod_N(\nu_k) \texttt{]}$.

Finally the whole local structure of a guest CA is encoded as $\texttt{[}\, bin(|Q|)\, cod(N)\, cod(f) \texttt{]}$.

3.2 Encodings of CA configurations

Given a guest CA G an encoding cod_Q of its states there are different possibilities to encode a G-configuration c. In order to avoid technical complications we will use the following.

A guest configuration $c \in Q^{\mathbb{Z}}$ is encoded by mapping each cell i to a *block* $b_i \in \{\texttt{0}, \texttt{1}, \texttt{[}, \texttt{]}\}^+$ which consists of three segments:

$$\langle \textit{block} \rangle = \texttt{[}\ \langle \textit{encoding segment} \rangle\ \langle \textit{state segment} \rangle\ \langle \textit{coin segment} \rangle\ \texttt{]}$$

The $\langle$*encoding segment*$\rangle$ will simply store the encoding of the guest CA. The $\langle$*state segment*$\rangle$ will store the encoding of the current state of one guest cell (and some additional data as explained later). The $\langle$*coin segment*$\rangle$ comprises two cells realizing an asynchronous coin.

The symbols of each block are stored in adjacent host cells and the blocks corresponding to consecutive guest cells are stored consecutively in the host.

3.3 Simulation

For the description of the simulation assume that the host is started in an initial configuration which is the encoding of a guest configuration as just described. The operation of the host will be explained using notions like “mark”, “signal” and “moving counter”. It is helpful to imagine that the local set of states of the host is subdivided into several *registers*. The complete array of cells then consists of several *tracks*; one contains the encoding of the guest configuration, while others are used for specific signals, counters, etc.

The construction/explanation of the host is successively refined in three steps:

- In Section 3.3.1 we review a standard construction for synchronous deterministic CA.
- In Section 3.3.2 the simple modification is added to get a simulator with asynchronous updating.
- In Section 3.3.3 we finally add the possibility to simulate guest which use asynchronous updating.

3.3.1 Synchronous Simulation of S-DCA

Parts of the following algorithm are very similar to the simulation described by Worsch and Nishio (2009).

The global steps of the guest are simulated one after the other. For each them the host proceeds as follows:

1. *Collect (the encodings of) the current states of the neighbors of the guest cell to be simulated.*

   For this signals have to be sent to neighboring blocks. The signals have to know how many blocks they have to travel (and they have to travel as many blocks back to their origin). One can use a standard signal of constant speed (smaller than 1 for the algorithms described below to work) and attach to it a pair of binary numbers (d, D), which initially are both the number of blocks the signal has to travel. The distance of the blocks is given by the encoded offsets of the guest neighborhood.

   When a signal arrives at the right block of a neighboring guest cell the encoding of its current states is copied and sent back to the origin.

2. *Use these information to select the corresponding rules of the guest transition function.*

   Upon arrival of a guest state in the block that had requested it, the corresponding rules of the transition table are marked as possibly relevant. If a state q_j could be obtained from neighbor j, state q_j in the local rule $[q_1, \ldots, q_k, q']$ is marked.

   The fact that all signals have returned a valid state can be recognized by the fact that in one local rule *all* $q_1, \ldots, q_k$ are marked. This is checked each time a state is marked.

3. *Update the state segment of the block.* The new state of the guest cell is read off the local rule $[q_1, \ldots, q_k, q']$ in which *all* of $q_1, \ldots, q_k$ have been marked and it is stored in the state segment.

   Also, all the marks attached to any rule in the encoding segment are removed.

3.3.2 Asynchronous Simulation of S-DCA

It is easy to realize the simulation of S-DCA on an asynchronous DCA. One just has to apply Nakamura's transformation to the CA described in the previous Section 3.3.1.

One should note, though, that one now has a different type of simulation. In the asynchronous host CA one can now have computations which are completely useless. As an extreme example consider the infinite computation where in each step only host cell 0 is active. On the other hand the computation during which in each step all host cells active is identical to the computation of the synchronous CA described in the previous subsection.

Assume that m steps of the synchronous host are necessary in order to simulate one step of the guest CA. Then those computations of the asynchronous host where each host cell makes progress (in the sense of Section 2.3.3) exactly m times correspond very closely to the m-step computations of the synchronous host.

The more general approach by Golze (1978) is to look at (equivalence classes of) space-time diagrams. If one is interested in the new states of only a finite number of guest cells, one can be even less restrictive. Questions of this type will be the topic of another paper.

3.3.3 Asynchronous Simulation of A-DCA

As a third step we now want to generalize the construction above in such a way that it is possible to simulate *asynchronous* guest DCA.

The goal now is to convert the asynchronous host DCA just described into a A-DCA H with the following property: For each initial configuration c' which is the encoding of a configuration c of an asynchronous guest DCA G, and for each computation of G starting with c there is a computation of H starting with c' which simulates the above.

The problem is that one has to exploit the nondeterminism inherent to the asynchronicity of H to systematically choose whether a guest cell should be passive (and its state left unchanged) or active (and simulated as described above).

We will now make use of the black boxes called asynchronous coin (which will be explained in detail in the next subsection. The behavior visible from outside is the following:

- Initially the two coin cells are in a waiting state w.
- A signal can be sent to the coin to the right, requesting a bit `0` or `1`.
- Both coin cells become "alerted" and after a certain number of steps when they have been active sufficiently often, the will have entered `00` or `11`.
- The cell to the left then has to change the request signal to a corresponding result signal which does not yet start to travel back.
- When the left coin cell (is active and) observes that the result bit has been copied it returns to the waiting state.
- Once the cell with the result signal observes this, said signal starts traveling back.

Assuming for the moment that this indeed can be implemented, it is used as follows for the simulation of asynchronous guests. Once all prerequisites $q_1, \ldots q_k$ of a local rule $[q_1, \ldots, q_k, q']$ have been marked, instead of assuming that the simulated cell is active and automatically updating the state segment, a request signal is sent to the coin. Only if a `1` is returned, the step of the guest cell is simulated. On the other hand a `0` is interpreted as indication that the simulated cell is passive. In this case the state segment is not changed; the marks at the local rules are removed nevertheless in preparation for the next possible step.

3.3.4 The "asynchronous coin"

It remains to explain how the asynchronous coin works. First we consider the somewhat easier case of purely asynchronous updating. An asynchronous coin consists of two adjacent cells. Instead of a detailed description of the local transition function Figure 1 captures the essence of the possible transitions of the pair. We use possible states w, a and b, and `0` and `1`.

Initially both coin cells are is state w, waiting for a request to produce a bit. For the figure we assume that the request comes from the left and that the result also will be consumed (carried away) to the left.

An edge indicates a possible transition from one pair of states to the next. The label of the edge shows which cells have to be active in order to realize this transition. 'L' means that only the left cell is active, 'R' that only the right cell is active and 'LR" that both cells are active. The trivial loops have been omitted.

The gray `ww` pair at the top of Figure 1 is the start, where the coin is waiting for a request to produce a `0` or `1`. When the (same) gray `ww` pair at the bottom has been reached, one tossing cycle has been finished. Edges which are straight lines indicates transitions which happen without exceptions. Starting from the top `ww` pair one can reach `00` as well as `11`. This is the result of the coin toss.

The dashed edges indicate that cells leave a state only under certain circumstances. The transition from ww to aw only happens when the left coin cell sees a request arriving from the left. And once the cells have entered states `00` or `11` they start to return to ww only when the left coin cell observes that its left neighbor has copied the produced bit.

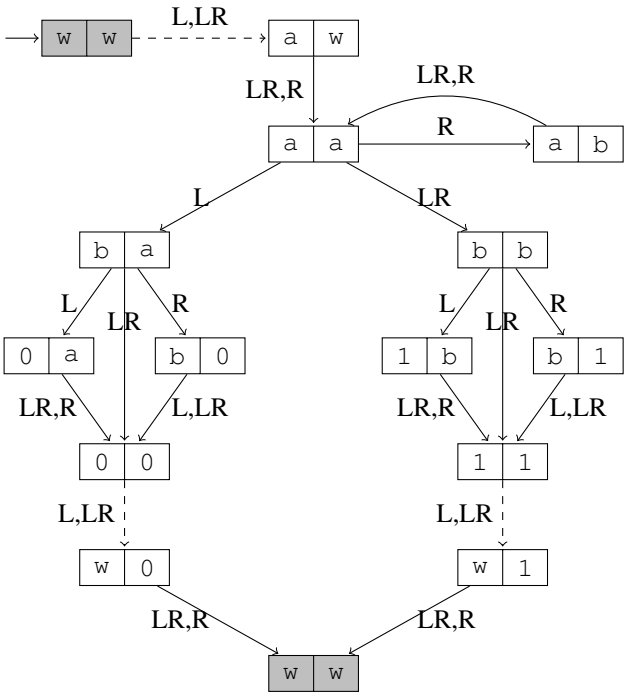


Fig. 1: The purely asynchronous coin.

There is one additional problem which has to be taken care of. In Section 3.3.2 the asynchronous host has been constructed from a synchronous one by applying Nakamura's technique.

But since the host is asynchronous it can happen that the asynchronous coins for neighboring guest cells produce their results after significantly different numbers of (host) steps. As a consequence it could happen that a guest state is requested (by a neighboring block) before it is computed. One possibility to avoid this is as follows: The mod 3 counter of the host cells is extended by an additional possible time stamp which is denoted by -1. It is interpreted as being "older" than each of the values 0, 1 and 2, so that

no activity can pass a cell with time stamp -1.

It suffices to slightly modify the behavior of the left coin cell. When it changes from state w to state a, it makes a backup of its time stamp and sets it to -1. The right coin cells copies this behavior. After both have returned to state w they reset their time stamp to the value backed up before.

4 Generalizations

4.1 Universality for α-asynchronous CA

One might have wondered why we did not choose a more symmetric approach in the previous section. Indeed one could exchange the roles of 'bb' and 'ab' at the top of Figure 1 without destroying the construction. But for the construction shown in Figure 2 for α-asynchronous CA this "asymmetry" is vital.

Our goal now is to describe an α-asynchronous host which is able to simulate all α-asynchronous guests; we want the same probability for guests and host. Remember that a cell is active with probability α and passive with probability $\beta = 1 - \alpha$.

Figure 2 results from Figure 1 by replacing the label 'L', 'R', 'LR', etc. with the corresponding probabilities. An edge labeled 'L' (or 'R') represented the case that exactly one of the two cells is active. In α-asynchronous CA this happens with probability $\alpha\beta$. An edge label-led 'LR' represented the case that both cells are active. In α-asynchronous CA this happens with probability α^2. An edge label-led 'L,LR' (or 'LR,R') represented the case that either exactly one of the two cells is active or both. In α-asynchronous CA this happens with probability $\alpha\beta + \alpha^2 = \alpha(\beta + \alpha) = \alpha$.

It is now easy to deduce the probabilities for the events that the pair of cells enters states `00` and `11` respectively. `00` will happen with probability $\alpha\beta/(\alpha\beta + \alpha^2) = \beta$, while `11` will happen with probability $\alpha^2/(\alpha\beta + \alpha^2) = \alpha$.

4.2 Universality for fully asynchronous CA

The construction of a fully asynchronous host which is able to simulate any fully asynchronous guest is beyond the scope of this paper. The problem is that the host has to ensure that for the simulation of one step of the guest exactly one guest cell is active. Currently the only possibility we are aware of is to use encodings which contain only one asynchronous coin. Thus one needs an encoding for configurations which is does not commute with the shift. Such an approach has been used in the literature (see e. g. Durand-Lose, 2000). But it is so different from the encodings used above, that we will not discuss this topic in this paper.

4.3 Asynchronous DCA which are universal for all asynchronous NCA

The host constructed in Section 3.3.3 can be generalized such that it can even simulate any asynchronous *non-deterministic* CA. We briefly sketch some modifications which are sufficient.

As the encoding of NCA one can choose the obvious generalization of the one described in Section 3.1: For each $q' \in f(q_1, \ldots, q_k)$ one adds `[` $cod_Q(q_1) \cdots cod_Q(q_k)\, cod_Q(q')$ `]` to encoding of the local transition function.

During the simulation of one global the situation may arise that at least two local rules are marked after the guest states from the guest neighbors have been retrieved. In such a situation

- first, the leftmost applicable local rule get a special mark.

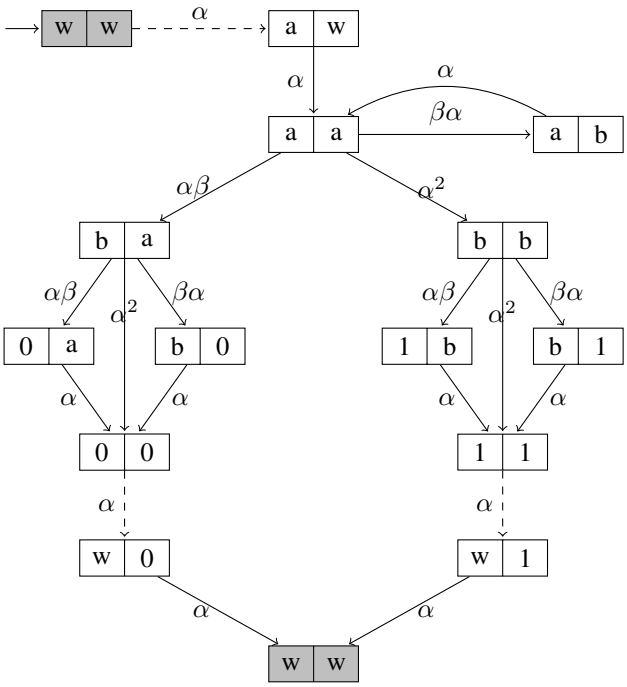


Fig. 2: The α-asynchronous coin. Missing probabilities belonging to self loops not shown for clarity.

- Then the asynchronous coin is used to produce a sequence of bits up to the first `1`. The arbitrary number of `0` generated before is used move the special mark from one applicable rule to the next one in a cyclic manner.

When the `1` appears, the local rule which has the special mark at that time is used for the transition.

5 Summary and Outlook

We have shown that there is one purely asynchronous DCA which can simulate all purely asynchronous DCA, and similarly for α-asynchronous DCA. A by-product was an asynchronous but "otherwise" deterministic CA that can simulate all asynchronous *nondeterministic* CA. This improves an earlier construction by Golze.

We have restricted ourselves to one-dimensional CA, but only in order keep the descriptions and notations a little bit simpler. The generalizations of the above results to higher dimensions are straight forward.

A similar result for fully asynchronous CA can be obtained but it seems to require a less stringent definition of simulation.

In our opinion one major interesting open problem is: Can a definition and concepts of intrinsic universality as proposed by Ollinger (2008); Delorme et al. (2010) be generalized to CA which are not synchronous and deterministic? It may be that the idea of U-V-simulation proposed by Golze (1978) is

useful in this context.

References

M. Delorme, J. Mazoyer, N. Ollinger, and G. Theyssier. Bulking II: Classifications of cellular automata. *CoRR*, abs/1001.5471, 2010.

J. O. Durand-Lose. Reversible space-time simulation of cellular automata. *Theoretical Computer Science*, 246(1-2):117–129, 2000.

N. Fatès and L. Gerin. Examples of fast and slow convergence of 2d asynchronous cellular systems. In H. Umeo and et al., editors, *Proceedings ACRI 2008*, pages 184–191, 2008.

U. Golze. (A-)synchronous (non-)deterministic cell spaces simulating each other. *Journal of Computer and System Sciences*, 17(2):176–193, 1978.

J. Lee, S. Adachi, F. Peper, and K. Morita. Asynchronous game of life. *Physica D*, 194(3-4):369–384, 2004.

K. Nakamura. Asynchronous cellular automata and their computational ability. *Systems, Conputers, Control*, 5(5):58–66, 1974.

N. Ollinger. Universalities in cellular automata; a (short) survey. In B. Durand, editor, *Proceedings JAC 2008*, pages 102–118, 2008.

D. Regnault, N. Schabanel, and E. Thierry. Progresses in the analysis of stochastic 2d cellular automata: A study of asynchronous 2d minority. *Theor. Comput. Sci.*, 410(47-49):4844–4855, 2009.

T. Worsch and H. Nishio. Achieving universality of CA by changing the neighborhood. *Journal of Cellular Automata*, 4(3):237–246, 2009.

Automata 2010 — 16th Intl. Workshop on CA and DCS — short papers 351–356

Undecidability of the Openness problem of multidimensional cellular automata

Charalampos Zinoviadis †

University of Turku, Department of Mathematics, Turku 20014, Finland

We prove that given a multidimensional cellular automaton, it is undecidable whether the transition function defined by it is open with respect to the standard topology. This is another difference between the properties of one-dimensional cellular automata and their multidimensional counterparts. The proof is based on a modification of Kari's original proof of the undecidability of the reversibility problem for multidimensional cellular automata.

Keywords: Cellular automata, dimension sensitive properties, openness, plane-filling property.

1 Preliminaries

We will use the abbreviation 1-D , 2-D , d-D for one-dimensional, two-dimensional and d-dimensional, respectively.

Let S be a finite set of *states* called the *alphabet*. A d-D *configuration* over S is a function $c\colon \mathbb{Z}^d \to S$ that assigns a state to every position of the d-D grid $\mathbb{Z}^d$. The set of all the d-D configurations $S^{\mathbb{Z}^d}$ is called the d-D *full shift*.

We can endow the space $S^{\mathbb{Z}^d}$ with a topology so that the resulting topological space is compact. The simplest way to describe this topology is by use of the *cylinders*

$$Cyl(D,p) = \{c \in S^{\mathbb{Z}^d} \colon c(\vec{n}) = p(\vec{n}),\ \text{for every } \vec{n} \in D\} \tag{1}$$

where D is a finite subset of $\mathbb{Z}^d$ and $p\colon D \to S$ assigns states only to the cells of D.

It can be easily seen that the cylinders satisfy the axioms for being a base of a topology.

Cellular automata (CA from now on) are defined formally as quadruples $A = (d, S, N, g)$, where

- $d \geq 1$ is the *dimension* of A,
- S is an alphabet called the *state set*,
- $N = (\vec{n_1}, \vec{n_2}, \ldots, \vec{n_m})$, where $\vec{n_i} \in \mathbb{Z}^d$ and $\vec{n_i} \neq \vec{n_j}$ for $i \neq j$ is the *neighborhood vector*, and

†This work has been supported by the Alexander S. Onassis Public Benefit Foundation and by the Academy of Finland grant 131558.

- $g\colon S^m \to S$ is the *local function*.

Here, d defines the dimension of the configurations on which A will work. For example, if $d = 1$ then the space on which A acts is $S^{\mathbb{Z}}$. The elements of the neighborhood vector specify the (ordered) relative locations of the neighbors of a cell: the neighbors of cell $\vec{n}$ are the cells $\vec{n}+N = (\vec{n}+\vec{n_1}, \vec{n}+\vec{n_2}, \ldots, \vec{n}+\vec{n_m})$.

In every time step, the local rule g is used to change a configuration c to another one c' in the following way:

$$c'(\vec{n}) = g(c(\vec{n}+N)) = g(c(\vec{n}+\vec{n_1}), c(\vec{n}+\vec{n_2}), \ldots, c(\vec{n}+\vec{n_m})) \tag{2}$$

The transformation $c \mapsto c'$ defines a global function

$$G\colon S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}, \tag{3}$$

the *transition function* of the CA. This is our main object of study. In fact, when we talk about a CA, we will often refer only to its transition function.

A CA is called *reversible* if it is bijective and its inverse function is also a CA. The following proposition is a classical result concerning reversible CA:

Proposition 1 *A CA is reversible if and only if it is injective.*

Wang tiles are unit squares with colored edges. A finite set T of tiles is called a *tile set.* A *tiling* with tiles from the tile set T is a function $c\colon \mathbb{Z}^2 \to T$. Intuitively, a tiling is a way to fill the plane with unit squares from T, where abutting squares are put side-to-side. Notice that we are not allowed to rotate the tiles. A tiling c is *valid at point* $(x, y) \in \mathbb{Z}^2$ if the edges of the tile $c(x, y)$ have the same color as the abutting edges of its neighboring tiles, i.e. if the upper edge of $c(x, y)$ has the same color as the lower edge of $c(x, y+1)$, the right edge of $c(x, y)$ has the same color as the left edge of $c(x+1, y)$ etc. A tiling c is called *valid* if it is valid at all points $(x, y) \in \mathbb{Z}^2$. We also use the expression that T *admits* the valid tiling c.

The following proposition states a fundamental fact about Wang tiles:

Proposition 2 (Compactness principle) *If a tile set can tile validly arbitrarily large squares, then it can tile validly the whole plane.*

Finally, let us introduce the decision problem from which we will do our reduction in Section 4:

- **Domino problem**
- **Input**: An arbitrary Wang tile set T.
- **Question**: Does T admit a valid tiling?

The following proposition is the single most famous result about Wang tiles:

Proposition 3 *(1; 2) The Domino problem is undecidable.*

2 The plane-filling property

Directed tiles are normal Wang tiles to which a *follower vector* $\vec{f} \in \mathbb{Z}^2$ is associated. A *directed tile set* is a set of directed tiles, i.e. a pair (T, F), where T is a Wang tile set and $F\colon S \to \mathbb{Z}^2$ is a function that assigns a follower vector to every tile. From now on, we will refer to a directed tile set using only its "base" tile set T. Let $c \in T^{\mathbb{Z}^2}$ be a tiling, which is not necessarily a valid tiling, and let $\vec{p} \in \mathbb{Z}^2$ be a position of the plane. The notion of validness of c in position $\vec{p}$ is the same as in the undirected case, which means that we do not care about follower vectors when we consider whether c is valid in $\vec{p}$ or not. The *follower of $\vec{p}$ in c* is the position $\vec{p} + F(c(\vec{p}))$. In other words, the follower is the cell to which the follower vector of the tile in position $\vec{p}$ is pointing to. Notice that in different tilings the same position might have different followers. However, we will usually talk about the follower of a position, assuming that the tiling to which we are referring is fixed. Also, observe that the notions of follower position and validness are independent. Otherwise stated, in a tiling $c \in T^{\mathbb{Z}^2}$ every position has a follower, not only those positions where c is valid. In the tile set we are going to use, the follower of every position is one of the four adjacent positions, that is $F(a) \in \{(\pm 1, 0), (0, \pm 1)\}$, for every $a \in T$.

A sequence $\vec{p_1}, \vec{p_2}, \ldots, \vec{p_k}$, where every $\vec{p_i} \in \mathbb{Z}^2$, is called a *path* on c if $\vec{p}_{i+1}$ is the follower of $\vec{p_i}$, for every $i = 1, 2, \ldots, k-1$. The notion of a *two-way infinite path* is defined analogously in the obvious way.

A directed tile set T is set to have the *plane-filling property* if it satisfies the following conditions:

1. T admits a valid tiling of the plane.
2. For every tiling $c \in T^{\mathbb{Z}^2}$, only two different types of infinite paths can appear:
   (a) There exists a position on the path where the tiling is not valid, or
   (b) the path covers arbitrarily large squares.

Therefore, if the tiling conditions are not violated on any position of the path, then for every $n \geq 1$, there exist an $n \times n$ square each tile of which is visited by the path and, hence the path must be infinite. Notice, also, that this condition does not claim anything about the validity of the whole configuration. As long as the configuration is valid on the path, arbitrarily large squares are visited. This does not prevent tiling errors from occuring outside the path.

The above definitions would have absolutely no meaning if not for the following proposition, whose proof will not be given.

Proposition 4 *(3) There exists a fixed directed tile set K with the plane-filling property.*

We will also need the following lemma:

Lemma 1 *(4) Let K be the tile set of Proposition 4. There exists a valid tiling k of K where all positions of the plane belong to the same path, i.e. they form a two-way infinite, non-intersecting, plane-filling path.*

3 Open CA

Let $G\colon A^{\mathbb{Z}^d} \to A^{\mathbb{Z}^d}$ be a CA. Then, G is called *open* if $G(W)$ is open for every open $W \subseteq S^{\mathbb{Z}^d}$, that is if G is an open function with respect to the standard topology.

Let F be a 1-D CA with state set S, neighborhood $\{0,1\}$ and local rule f. We say that F is *left-permutive* if for every $b, c \in S$, there exists a unique $a \in S$ such that $f(a,b) = c$. This also means that if $a_1 \neq a_2$ and $f(a_1,b_1) = f(a_2,b_2)$, then $b_1 \neq b_2$.

Two configurations $c_1, c_2 \in S^{\mathbb{Z}}$ are called *left-asymptotic* if there exists $m_0 \in \mathbb{Z}$ such that $c_1(m) = c_2(m)$, for every $m \leq m_0$. *Right asymptotic* configurations are analogously defined. A 1-D CA F is called *right-closing* if $F(c_1) \neq F(c_2)$, for every pair of left-asymptotic, non-equal configurations c_1, c_2. The definition of *left-closing* 1-D CA can be given in a similar way. It is easy to see that a left-permutive CA is also left-closing.

For the 1-D case, there exists the following characterization of open CA:

Proposition 5 *(5) A 1-D CA is open if and only if it is both right- and left-closing.*

Corollary 1 *There exists a 1-D CA F that is left-permutive but not open.*

Proof: Let $F\colon S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ be the 1-D CA with the neighborhood $\{0,1\}$ and the following local rule:

x \ y	0	1	2
0	0	1	1
1	1	0	2
2	2	2	0

F is left-permutive as every column of its transition matrix is a permutation of the state set. However, as $F({}^{\omega}0.1^{\omega}) = F({}^{\omega}0.2^{\omega}) = {}^{\omega}01.0^{\omega}$, F is not right-closing. According to Proposition 5, F is not open. □

4 Undecidability of the openness problem

In this final section, we will prove that the following decision problem is undecidable:

- **Openness problem of d-D CA**, $d \geq 2$.
- **Input**: An arbitrary d-D CA G.
- **Question**: Is G open?

The respective problem for 1-D CA is known to be decidable, see (6). In fact, using methods similat to the ones introduced in (8), a simple polynomial time algorithm can be given for testing right- and left-closingness. According to Proposition 5, this gives a polynomial time algorithm for testing openness of 1-D CA.

It is easy to see that it is enough to prove the result for $d = 2$.

Proposition 6 *The Openness problem of 2-D CA is undecidable.*

Proof: We are going to reduce the Domino problem to the Opennes problem of 2-D CA. Given an arbitrary tile set T, we algorithmically construct the following 2-D CA G: The state set is $S = K \times T \times \{0, 1, 2\}$, where K is the fixed tile set from Proposition 4. Therefore, the CA is working on configurations consisting of three different layers. In the first layer, there exist tiles from the fixed tile set K; in the second layer there exist tiles from the arbitrary tile set T, and on the third one there are the letters 0,1 and 2. Let us call these layers K-layer, D-layer and letter-layer repsectively. The neighborhood used is $\{(0, 0), (0, 1), (1, 0), (0, -1), (-1, 0)\}$, so that every cell looks at the state of itself and those of its immediate neighbors in order to determine its state in the next time step. The local rule only updates the letter components of a position $\vec{p}$ according to the following rule:

- If either the T-layer or the K-layer contains a tiling error at $\vec{p}$, then the letter is not changed, but
- if the tiling is valid in both the K- and T-layers in $\vec{p}$, then the letter of position $\vec{p}$ is changed to $f(a, b)$, where f is the local rule of the 1-D CA defined in Corollary 1, a is the current letter in position $\vec{p}$ and b is the letter of the follower of $\vec{p}$.

Let us now prove that this CA is not open if and only if T admits a valid tiling. This reduces the Domino problem to the Openness problem of 2-D CA and, hence, completes the proof.

Suppose that T admits a valid tiling t. Arguing by contradiction, assume that G is open. Let k be the valid K-tiling from Lemma 1 and consider the set $B = k \times t \times \{0, 1, 2\}^{\mathbb{Z}^2}$. Since in k there exists only one path and both k and t are valid everywhere, the restriction of G on B is in some sense the same as the 1-D CA F. Indeed, the unique path in k defines a homeomorphism $\phi\colon \mathbb{Z}^2 \to \mathbb{Z}$ that preserves open sets and in addition $G(k, t, c) = (k, t, F(\phi(c)))$. Hence, identifying $W \subseteq \mathbb{Z}^2$ with $\phi(W) \subseteq \mathbb{Z}$, we have that $G^{-1}(B) = B$ and $G(k, t, W) = (k, t, F(W))$, for every $W \subseteq \{0, 1, 2\}^{\mathbb{Z}^2}$. According to Theorem 1,page 116 in (7), if $W \subseteq \{0, 1, 2\}^{\mathbb{Z}^2}$ is open, then $G(k, t, W) = (k, t, F(W))$ is also open in the relative topology. In addition, since it is a basic topological fact that all projections are open, $F(W)$ is also open. But this means that F is an open CA, which is a contradiction. Therefore, G is not open.

Suppose, then, that G is not open. Since reversible CA are always open, G is not reversible either. Using Proposition 1, we can conclude that G is not injective. This means that there exist two different configuration c_1 and c_2 such that $G(c_1) = G(c_2)$. The tile components are not changed by G and thus they must be identical in c_0 and c_1, so there exists a position $\vec{p_1}$ where they have different letters. Using the left-permutativity of f and the fact that after the application of G these letters become identical, we

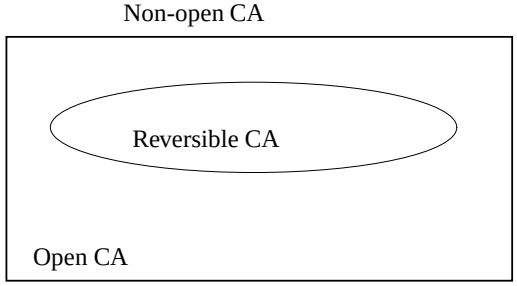


Fig. 1: Reversible CAs are recursively inseparable from non-open CAs.

can see that the configurations are valid in both layers and the letters in the follower $\vec{p_2}$ of $\vec{p_1}$ must also be different. Repeating this reasoning, we obtain an infinite path $\vec{p_1}, \vec{p_2}, \vec{p_3}, \ldots$ such that the tiling conditions are satisfied in both the K- and T- components at all positions of the path. Since K has the plane-filling property, this infinite path covers arbitrarily large squares. Therefore, T can tile validly arbitrarily large squares, which means that it admits a valid tiling of the whole plane, according to Proposition 2. □

By looking a little closer to the CA G of the previous proof, we can see that it is reversible if and only if it is open. Hence, we have actually proven something stronger than the claim of Proposition 6, namely that the class of reversible 2-D CA is recursively inseparable from the class of non-open ones.

References

[1] Berger, Robert, The undecidability of the domino problem, Memoirs of the AMS, Vol. 66, 1966, pp. 1-72.

[2] Robinson, Raphael M., Undecidability and nonperiodicity for tilings of the plane, Inventiones Mathematicae, Vol.12, 1971, pp. 177-209.

[3] Kari, Jarkko, Reversibility and Surjectivity Problems of Cellular Automata, Journal Computer Systems Science, Vol. 48, 1994, pp. 149-182.

[4] Meyerovitch, Tom, Finite entropy for multidimensional cellular automata, Ergodic Theory and Dynamical Systems, Vol. 28, 2008, pp. 1243-1260.

[5] Kurka, Petr, Topological and symbolic dynamics, Societe Mathematique de France, 2003, Cours Specialises 11.

[6] Wilson, Stephen J., Decision procedures for openness and local injectivity, Complex Systems, Vol.5, 1991, pp. 497-508.

[7] Kuratowski, Kazimierz, Topology- Volume 1, Academic Press, 1966.

[8] Sutner, Klaus, De Bruijn graphs and linear cellular automata, Complex Systems, Vol. 5, 1991, pp. 19-30.

Organisation:

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche NANCY - GRAND EST

Local Sponsors:

Loria

cnrs

dépasser les frontières

La Région Lorraine

ville de Nancy

Nancy-Université Université Nancy 2

Nancy-Université Université Henri Poincaré

Nancy-Université INPL

Grand Nancy communauté urbaine

International Sponsors:

ESMTB

ifip International Federation for Information Processing